Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FIN DE GRADO

Título: Automatización del diseño de las superficies limitadoras de obstáculos

Titulación: Grado en Ingeniería de Aeropuertos

Autor: Wenhao Cheng

Director: Óscar Maldonado Díaz

Data: 30 de Junio de 2015

Título: Automatización del diseño de las superficies limitadoras de obstáculos

Autor: Wenhao Cheng

Director: Óscar Maldonado Díaz

Data: 30 de Junio de 2015

Resumen

Actualmente, la seguridad aérea es uno de los temas más sensibles que hay en el ámbito aeronáutico. A pesar de que la relación victimas/accidentes en las operaciones aéreas es bastante menor respecto a lo que ocurre en otros medios, como por ejemplo el coche, el impacto económico y mediático que genera un accidente aéreo es de mucho mayor envergadura. Por lo cual, las organizaciones internacionales y locales han creado un conjunto de regulaciones con el objetivo de reducir y evitar las posibles causas que pudieran derivar en catástrofes.

Uno de los principales temas que trata dichas regulaciones, son las servidumbres aeronáuticas, el objetivo de estas servidumbres es la de proteger el espacio aéreo de las instalaciones del aeropuerto y sus proximidades, aplicando una serie de restricciones que prohíben la construcción o presencia de cualquier objeto, que vulnere la seguridad operacional de las aeronaves.

En este proyecto, se desarrolla una aplicación a partir de la cual podremos obtener el diseño de las servidumbres de un aeropuerto de acuerdo con las regulaciones y las normas de la OACI como del BOE. Por lo cual, es necesario analizar y comprender dichas normativas.

La aplicación es diseñada en lenguaje C# mediante *Microsoft Visual Studio 2012*. A partir de la cual, se obtiene la interfaz con el que el usuario puede interactuar.

Finalmente, también es necesario asimilar e implementar en el código de la aplicación, los comandos y las bibliotecas de enlace dinámico (.dll), para poder representar las servidumbres en un entorno *AUTOCAD*.

Title: Automatización del diseño de las superficies limitadoras de obstáculos

Author: Wenhao Cheng

Director: Óscar Maldonado Díaz

Date: June, 30th 2015

Abstract

Air safety is one of the most sensitive issues in the fascinating world of aeronautics. Even though the rate of victims to accident in case of any aircraft catastrophe is relatively low in comparison with other means of transport such as a car, the echo of the news and its economic impact is far away high with respect to the others. Because of that, the international and local organizations of aviation have created a set of regulations in order to reduce and prevent any possible causes that could lead to any kind of disaster.

Aeronautical servitude with an objective of protecting the airspace of airport facilities and its vicinity, applying a series of restrictions that ban construction or presence of any object that infringe aircrafts operation safety are the principal terms of the cited regulations.

This document is development of an application from which airports servitude is designed in accordance with ICAO's and BOE's regulations and standards that leads to analyse and understand these regulations.

The application is designed in the programming language C# using Microsoft Visual Studio 2012 as an interface for the user interaction.

Finally, it is necessary to assimilate and implement the commands and .dll libraries in applications's code in order to be able to represent servitudes in the AUTOCAD environment.

Contenido

Introducción 1
Capítulo 1.Servidumbres aeronáuticas 2
1.1 Servidumbres físicas 2
1.2 Servidumbres radioeléctricas 7
1.2.1 Servidumbres radioeléctricas-ILS 9
1.3 Servidumbres aeronáuticas y seguridad aérea 11
Capítulo 2. Desarrollo de la aplicación 14
2.1 Descripción general de la aplicación 14
2.2 Arquitectura de la aplicación 15
2.3 Programas utilizados 15
2.4 Requisitos necesarios 15
2.5 Lenguaje de programación 16
Capítulo 3. Solución adoptada 17
3.1 Determinación de las intersecciones 17
3.2 Representación de las servidumbres 22
3.3 Procedimiento de análisis y obtención de parámetros 22
3.3.1 Horizontal interna y Cónica 23
3.3.2 Aproximación y Transición 26
3.3.3 Superficie de aterrizaje interrumpido y Superficie de ascenso en el despegue 31
3.3.4 Aproximación interna y Transición interna 34
3.3.5 Servidumbres radioeléctricas 37
3.3.5.1 Servidumbre radioeléctrica ILS 40
3.4 Traslación de los puntos 43
3.5 Procesamiento de los parámetros de entrada 46
3.6 Compatibilidad de datos 47
3.7 Introducción y modificación de la lista de obstáculos y la lista de instalaciones radioeléctricas. 48
4. Conclusiones 49
5. Bibliografía 51

Índice de figuras

Figura 1.1 Dimensiones y pendientes de las superficies limitadoras de obstáculos para pistas destinadas al aterrizaje……3
Figura 1.2 Dimensiones y pendientes de las superficies limitadoras de obstáculos para pistas destinadas al despegue……4
Figura 1.3 Vistas en planta y en sección de varias superficies limitadoras de obstáculos……5
Figura 1.4 Vistas en planta y en sección de las superficies de transición interna, aproximación interna y de aterrizaje interrumpido……6
Figura 1.5 Servidumbres del ILS/LOC……..10
Figura 1.6 Servidumbres del ILS/GP……11
Figura 1.7 Detalle de la noticia sobre el accidente aéreo ocurrido en el aeropuerto de Sao Paulo, 2007……....12
Figura 1.8 Detalle de la noticia sobre el accidente aéreo ocurrido en el aeródromo de Sant Quirze del Vallés, 2005……13
Figura 2.1 Librerías a referenciar si se trabaja con AUTOCAD 2015……16
Figura 2.2 Directivas using a aplicar……16
Figura 3.1 Sectorización de la superficie de aproximación y aproximación interna..……18
Figura 3.2 Tipos de sectores……..19
Figura 3.3 Sector triangular donde la y crece a medida que la x decrece……..20
Figura 3.4 Sector triangular donde la y decrece a medida que la x decrece….21
Figura 3.5 Paleta de colores con las que se representaran las servidumbres……22
Figura 3.6 Límites superficiales de la Horizontal interna y la Cónica……..23
Figura 3.7 Límites superficiales de la Aproximación y la Transición……...26
Figura 3.8 Límites superficiales de la Superficie de aterrizaje interrumpido y la Superficie de ascenso en el despegue……31
Figura 3.9 Límites superficiales de la Aproximación interna y la Transición interna……34
Figura 3.10 Límites superficiales de una servidumbre radioeléctrica……39
Figura 3.11 Límites superficiales de una servidumbre radioeléctrica ILS/LOC.41
Figura 3.12 Límites superficiales de una servidumbre radioeléctrica ILS/GP...42
Figura 3.13 Orientación respecto al norte magnético……43
Figura 3.14 Determinación de ángulos para pistas no horizontales……...44
Figura 3.15 Representación de servidumbres con las mismas características dimensionales, pero con diferente orientación……...45
Figura 3.16 Mensajes de advertencia que el usuario verá en caso de que no se haya rellenado correctamente los parámetros……...46
Figura 3.17 Mensaje de advertencia que el usuario verá en caso de que se intente ejecutar alguna función sin haber introducido previamente los parámetros……46
Figura 3.18 Mensaje de advertencia que el usuario verá en caso incompatibilidad (izquierda); Incompatibilidades de diseño de pista (derecha).47
Figura 3.19 Mensaje de advertencia que el usuario verá en caso incompatibilidad (izquierda); Incompatibilidades de aproximación (derecha)…47
Figura 3.20 Mensaje de advertencia que indica duplicación de identificadores……...48

Índice de tablas

Tabla 1.1. Clasificación de pista según su longitud y ancho……..2
Tabla 1.2. Dimensiones y pendientes de las servidumbres radioeléctricas…….8

Introducción

Actualmente, debido a la gran repercusión genera un accidente aéreo tanto a nivel económico como en la pérdida de vidas humanas. Es necesario reducir las probabilidades de que tengan lugar los accidentes, y sus posteriores consecuencias. Por ello, es muy importante seguir las regulaciones que las autoridades internacionales (OACI) y locales (AESA) dictaminan respecto a la seguridad aérea durante el diseño de los aeropuertos y aeródromos.

Una de las principales cuestiones que abordan estas regulaciones son las servidumbres aeronáuticas. La finalidad de estas servidumbres es la de proteger el espacio aéreo de los aeropuertos y sus proximidades. A partir de la instauración de un conjunto de restricciones, que prohíben la construcción o la presencia de cualquier edificio u objeto, que vulnere la seguridad operacional de las aeronaves.

El objetivo de este proyecto, es desarrollar una aplicación con la cual podremos obtener el diseño de las servidumbres de un aeropuerto de acuerdo, con las regulaciones y las normas del *Anexo 14 de la OACI* y del *Decreto 584/1972 de 24 de Febrero de la BOE*. Así como también, identificar la existencia de los obstáculos que infrinjan alguna de las restricciones impuestas.

Por lo cual, de acuerdo con las especificaciones del Anexo 14 de la OACI, se determinan las características dimensionales de las servidumbres físicas. Por otro lado, a partir de las disposiciones del Decreto 584/1972 de 24 de Febrero de la BOE, se obtienen las características dimensionales de las servidumbres radioeléctricas. [1] [2]

Una vez se haya comprendido como se extienden las servidumbre alrededor de la infraestructura aeroportuaria, se procede al desarrollo del programa mediante *Microsoft Visual Studio 2012* en lenguaje C#. A partir de la cual, se diseña la interfaz con el que el usuario puede introducir los parámetros necesarios y visualizar los resultados.

Finalmente, es necesario asimilar e implementar en el código de la aplicación, los comandos y las bibliotecas de enlace dinámico (.dll), para poder representar las servidumbres en un entorno *AUTOCAD*, permitiendo así al usuario visualizar el resultado y guardarlo como un plano en formato *.dwg*.

Capítulo 1.Servidumbres aeronáuticas

Se define como servidumbre aeronáutica. Toda restricción que se implemente con el objeto de asegurar el espacio aéreo, de forma que se puedan realizar con seguridad, las operaciones de las aeronaves alrededor de la zona de influencia de la infraestructura aeroportuaria.

Además, también se consideran servidumbres aeronáuticas, aquellas que existan con el fin de que no se generen interferencias, que dificulten las comunicaciones entre los instrumentos de navegación, ayudas a la aproximación y las aeronaves.

De acuerdo con esta definición, podemos clasificar las servidumbres como:

- Servidumbres físicas.
- Servidumbres radioeléctricas.

1.1 Servidumbres físicas

Las constituyen aquellas, que se establecen en los aeródromos y sus alrededores, con el fin de garantizar la seguridad en el movimiento de las aeronaves durante las operaciones, por lo cual, su principal función es evitar la proliferación de nuevos obstáculos en las proximidades del aeropuerto.

Con este objetivo, se definen las Superficies Limitadoras de Obstáculos (SLO), superficies imaginarias situadas en el entorno aeroportuario, que mantienen el espacio aéreo libre de obstáculos, fijando los límites hasta donde los objetos pueden proyectarse en el espacio aéreo.

Las características dimensionales de las SLO, se clasifican según la clave de la pista, que viene determinado por la longitud de la pista, en el caso de las superficies relativas al aterrizaje también se tiene en cuenta el tipo de aproximación para la cual ha sido diseñada la pista.

Tabla 1.1. Clasificación de pista según su longitud y ancho. [1]

Núm. de clave	Longitud de campo de referencia del aeródromo	Anchura de la pista según la letra de clave					
		A	B	C	D	E	F
1	Menos de 800 m	18m	18m	23m	---	---	---
2	Desde 800 m hasta 1200 m (exclusive)	23m	23m	30m	---	---	---
3	Desde 1200 m hasta 1800 m (exclusive)	30m	30m	30m	45m	---	---
4	Desde 1800 m en adelante	---	---	45m	45m	45m	60m

En las siguientes figuras, se pueden apreciar los valores de las dimensiones que adoptan las diversas SLO tanto para pistas de aterrizaje como de despegue.

	Aproximación visual				Aproximación que no sea de precisión			Aproximación de precisión Categoría I		Aproximación de precisión Categoría II o III
	Número de clave				Número de clave			Número de clave		Número de clave
Superficies y dimensiones[a]	1	2	3	4	1,2	3	4	1,2	3,4	3,4
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
CÓNICA										
Pendiente	5%	5%	5%	5%	5%	5%	5%	5%	5%	5%
Altura	35 m	55 m	75 m	100 m	60 m	75 m	100 m	60 m	100 m	100 m
HORIZONTAL INTERNA										
Altura	45 m	45 m	45 m	45 m	45 m	45 m	45 m	45 m	45 m	45 m
Radio	2 000 m	2 500 m	4 000 m	4 000 m	3 500 m	4 000 m	4 000 m	3 500 m	4 000 m	4 000 m
APROXIMACIÓN INTERNA										
Anchura	—	—	—	—	—	—	—	90 m	120 m[e]	120 m[e]
Distancia desde el umbral	—	—	—	—	—	—	—	60 m	60 m	60 m
Longitud	—	—	—	—	—	—	—	900 m	900 m	900 m
Pendiente								2,5%	2%	2%
APROXIMACIÓN										
Longitud del borde interior	60 m	80 m	150 m	150 m	150 m	300 m	300 m	150 m	300 m	300 m
Distancia desde el umbral	30 m	60 m	60 m	60 m	60 m	60 m	60 m	60 m	60 m	60 m
Divergencia (a cada lado)	10%	10%	10%	10%	15%	15%	15%	15%	15%	15%
Primera sección										
Longitud	1 600 m	2 500 m	3 000 m	3 000 m	2 500 m	3 000 m	3 000 m	3 000 m	3 000 m	3 000 m
Pendiente	5%	4%	3,33%	2,5%	3,33%	2%	2%	2,5%	2%	2%
Segunda sección										
Longitud	—	—	—	—	—	3 600 m[b]	3 600 m[b]	12 000 m	3 600 m[b]	3 600 m[b]
Pendiente	—	—	—	—	—	2,5%	2,5%	3%	2,5%	2,5%
Sección horizontal										
Longitud	—	—	—	—	—	8 400 m[b]	8 400 m[b]	—	8 400 m[b]	8 400 m[b]
Longitud total	—	—	—	—	—	15 000 m	15 000 m	15 000 m	15 000 m	15 000 m
DE TRANSICIÓN										
Pendiente	20%	20%	14,3%	14,3%	20%	14,3%	14,3%	14,3%	14,3%	14,3%
DE TRANSICIÓN INTERNA										
Pendiente	—	—	—	—	—	—	—	40%	33,3%	33,3%
SUPERFICIE DE ATERRIZAJE INTERRUMPIDO										
Longitud del borde interior	—	—	—	—	—	—	—	90 m	120 m[e]	120 m[e]
Distancia desde el umbral	—	—	—	—	—	—	—	[c]	1 800 m[d]	1 800 m[d]
Divergencia (a cada lado)	—	—	—	—	—	—	—	10%	10%	10%
Pendiente	—	—	—	—	—	—	—	4%	3,33%	3,33%

Figura 1.1 Dimensiones y pendientes de las superficies limitadoras de obstáculos para pistas destinadas al aterrizaje. [1]

Superficie y dimensiones[a]	Número de clave		
	1	2	3 ó 4
(1)	(2)	(3)	(4)
DE ASCENSO EN EL DESPEGUE			
Longitud del borde interior	60 m	80 m	180 m
Distancia desde el extremo de la pista[b]	30 m	60 m	60 m
Divergencia (a cada lado)	10%	10%	12,5%
Anchura final	380 m	580 m	1 200 m 1 800 m[c]
Longitud	1 600 m	2 500 m	15 000 m
Pendiente	5%	4%	2%[d]

Figura 1.2 Dimensiones y pendientes de las superficies limitadoras de obstáculos para pistas destinadas al despegue. [1]

Es importante tener en cuenta las siguientes consideraciones: [1]

-En las Superficies de las pistas de aterrizaje:

a. Salvo que se indique de otro modo, todas las dimensiones se miden horizontalmente.
b. La longitud de la sección horizontal de la superficie de aproximación debe empezar en el punto donde se produzca la intersección donde la pendiente es de 2,5% a una altura de 150m (valor mínimo de la OCA/H). En caso de que se modifique dicha altura será necesario recalcular el valor de la longitud de la sección horizontal para que asegure la protección.
c. La Superficie de aterrizaje interrumpido comenzará desde el extremo de la franja, en caso de pistas de clave 1 o 2 de pistas diseñadas para una aproximación de precisión de categoría I.
d. Para el resto de casos, la Superficie de aterrizaje interrumpido empezará en el extremo de la pista en caso de que la longitud de la pista sea menor a la distancia especificada.
e. Para pistas con letra de clave F, se deberá incrementar el valor de la longitud del borde interior de la Superficie de aterrizaje interrumpido a 155m.

-En la Superficie de ascenso en el despegue:

a. Salvo que se indique de otro modo, todas las dimensiones se miden horizontalmente.

b. La Superficie de ascenso en el despegue comienza en el extremo de la zona libre de obstáculos si la longitud excede de la distancia especificada.
c. El valor será de 1800m cuando la derrota prevista incluya cambios de rumbo mayores de 15º en las operaciones realizadas en IMC, o en VMC durante la noche.
d. En caso de que las características operacionales y permitan una reducción de la pendiente, se debe ajustar el valor de la longitud de la Superficie en el despegue para que pueda proporcionar protección hasta una altura de 300 m.

De tal manera, de acuerdo con el Anexo 14 de la OACI existen las siguientes SLO:

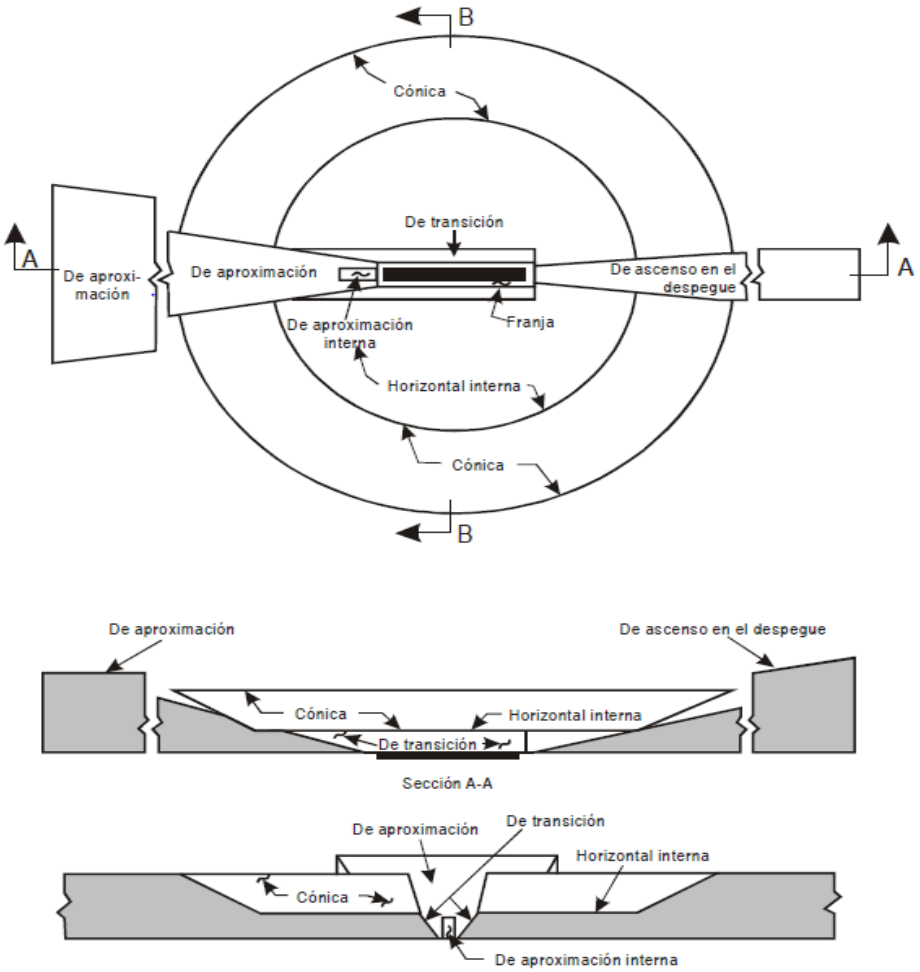

Figura 1.3 Vistas en planta y en sección de varias superficies limitadoras de obstáculos.

- Superficie Horizontal interna: La finalidad de esta superficie, es proteger el espacio aéreo para el circuito visual dentro del cual la aeronave deba volar antes de aterrizar. Esta superficie se situará en un plano horizontal a una altura de 45m sobre el punto de referencia establecido por el aeródromo. Y, a partir de ahí, se extiende en cada cabecera de la pista un determinado valor de radio.
- Superficie cónica: Superficie de pendiente ascendente, que se extiende desde el perímetro de la superficie horizontal interna hasta una determinada altura sobre la superficie horizontal interna. Al igual que la superficie cónica, su función es la de proteger el espacio aéreo correspondiente al circuito previo al aterrizaje.
- Superficie de aproximación: Conjuntamente con las superficies de transición, define el espacio aéreo que debería mantenerse libre de obstáculos para proteger las aeronaves durante las maniobras de aproximación para el aterrizaje. Está constituido por una sucesión de planos, inclinados y horizontales. Los valores de las pendientes y las longitudes de éstos, están especificados en la *Figura 1.1*.
- Superficie de transición: La misión de esta superficie es proteger los edificios del recinto aeroportuario y sus proximidades, así como también, el equipamiento y las aeronaves estacionadas. Esta superficie, se extiende a lo largo de la franja y parte del borde de la superficie de aproximación y desde ahí hacia fuera con pendiente ascendente hasta la superficie horizontal interna.
- Superficie de aproximación interna: Constituida por una porción rectangular que se extiende antes del umbral. Además de las funciones de la Superficie de aproximación, la Superficie de aproximación interna, conjuntamente con las superficies de transición interna y de aterrizaje interrumpido, también se encarga de definir la zona libre de obstáculos en la parte del espacio aéreo inmediato de las pistas para aproximaciones de precisión.

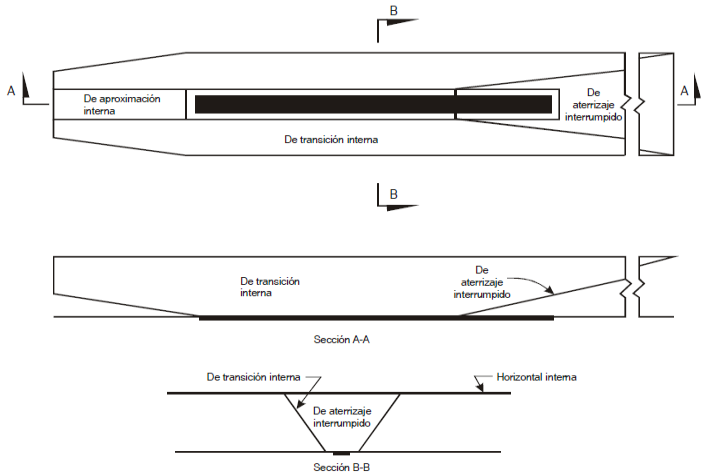

Figura 1.4 Vistas en planta y en sección de las superficies de transición interna, aproximación interna y de aterrizaje interrumpido.

- Superficie de transición interna: Es una superficie con pendiente ascendente, que se apoya en el borde la superficie de aproximación interna a lo largo de la franja paralela a la pista y en el borde de la superficie de aterrizaje interrumpido; y desde ahí hasta la superficie horizontal interna. Esta superficie se define con el objetivo de proteger las infraestructuras de ayuda a la navegación, las aeronaves y los vehículos que estén en las cercanías de la pista.
- Superficie de aterrizaje interrumpido: Su misión es garantizar que se puedan llevar a cabo de manera segura las maniobras de aproximación frustrada, realizadas por debajo de la altura de decisión. Está formado por un plano situado a una distancia específica después del umbral, que se extiende entre las dos superficies de transición.
- Superficie de ascenso en el despegue: Se implementa con el objeto de proteger los aviones durante el despegue. Consta de un plano inclinado u otra superficie especificada, situada más allá del extremo de pista o de la zona libre de obstáculos.

1.2 Servidumbres radioeléctricas

Las superficies radioeléctricas, se implementan con el objetivo de evitar la aparición de nuevos obstáculos, que puedan provocar un descenso en la calidad de la señal transmitida por los sistemas de radio ayudas o sistemas de comunicaciones. Asegurando de tal manera, que la calidad de las señales de las comunicaciones permanezcan dentro de los límites requeridos.

A diferencia de las servidumbres físicas o SLO, que están perfectamente definidas en el Anexo 14 de la OACI, las servidumbres radioeléctricas no lo están. Por otro lado, en el Anexo 10 de la OACI están definidas las generalidades de las instalaciones radioeléctricas pero éstas deberán particularizarse dependiendo de las características de la radio ayuda (modelo, marca, radiación, calibración, etc.).

De hecho, las servidumbres radioeléctricas son objeto de asignaturas y textos específicos. En España, las servidumbres radioeléctricas vienen establecidas en el Decreto 584/1972 de 24 de Febrero. En dicho Decreto se definen las siguientes especificaciones para las superficies radioeléctricas. [2] [3]

- Instalaciones radioeléctricas aeronáuticas: Se considera como instalación radioeléctrica aeronáutica, todo equipo radioeléctrico (transmisor, receptor, reflector activo o pasivo), así como también, sus antenas, líneas de transmisión, sistemas de tierra y las construcciones que los contengan, sustenten o protejan, dependientes del Ministerio del Aire e instalados para establecer una transferencia de información, por medios radioeléctricos, entre puntos específicos, fijos o móviles.

- Zona de instalación: Superficie de terreno o agua, en el que están situados los elementos de una instalación aeronáutica.
- Punto de referencia de la instalación: Punto definido por las coordenadas geográficas en las cuales estará situado la instalación.
- Plano de referencia de la instalación: Plano horizontal que contiene el punto de referencia.
- Perturbaciones radioeléctricas: Son consideradas perturbaciones radioeléctricas a una instalación radioeléctrica aeronáutica, todas aquellas que son producidas por:

    - Absorciones y/o reflexiones de las ondas electromagnéticas radiadas o recibidas por la instalación.
    - Otras radiaciones ajenas a la misma.

- De tal manera, con la finalidad de reducir las perturbaciones, se imponen las siguientes servidumbres:

    - Zona de seguridad: Superficie de terreno o agua que rodea la instalación, y en la cual no se puede emplazar ninguna otra instalación, así como tampoco, la modificación de la instalación radioeléctrica sin previo consentimiento del Ministerio del Aire.
    - Zona de limitación de alturas: Área de terreno o agua que rodea el perímetro de la Zona de seguridad y que se extiende desde ahí hasta una determinada distancia.
    - Superficie de limitación de alturas: Superficie inclinada que parte del perímetro de la zona de seguridad sobre el plano de referencia, manteniendo con ésta una pendiente determinada, extendiéndose hasta el límite de la zona de limitación de alturas. Se prohíbe que ningún elemento sobre el terreno sobrepase en altura dicha superficie.

Por lo cual, los valores de las características dimensionales de las servidumbres radioeléctricas dependen del tipo de instalación. A continuación, en la siguiente tabla se pueden apreciar dichos valores:

Tabla 1.2. Dimensiones y pendientes de las servidumbres radioeléctricas. [2]

Instalación	Zona de seguridad - Metros	Zona de limitación de alturas - Metros	Sup. Limitación de alturas - Pendiente%
Centro de emisiones o receptores:			
Frecuencias bajas (LF) o	200	2000	10

medias (MF)			
Frecuencias altas (HF)	300	2000	7,5
Frecuencias muy altas (VHF) o ultra elevada (UHF)	300	2000	5
Radiobaliza marcadora de tipo “Z” (75MHz)	200	1000	100
Radiobaliza marcadora en abanico (“Fan Maker”) (75MHz)	200	1000	100
Radiofaros no direccionales	300	2000	10
Radiofaro omnidireccional VFH (VOR), equipo medidor de distancia (DME) y TACAN	300	3000	3
Radiogoniómetro VHF (VDF) o UHF (UDF)	300	5000	2
Radar de vigilancia primario o secundario (SSR)	300	5000	2

1.2.1 Servidumbres radioeléctricas-ILS

A diferencia de las otras instalaciones, el ILS (Sistema de Aterrizaje Instrumental) tendrá 2 superficies radioeléctricas 1 para el LOC (Localizador) y otro para el GP (Equipo de trayectoria de Planeo). [2]

- Localizador del ILS (LOC/ILS)
    - Zona de seguridad: La zona de seguridad del LOC/ILS viene definida por las intersecciones con el terreno de los cuatro planos verticales siguientes:
        - a) El perpendicular al vertical que contiene el eje de la pista y que pasa por el umbral de la pista más próxima al punto de referencia de la instalación.
        - El paralelo al a), a igual distancia del punto de referencia y al otro lado del mismo.
        - y d) Los vértices paralelos al eje de pista que pasan por las intersecciones de los a) y b) con otros dos planos verticales d) y f) que pasan por el punto de referencia y forman un ángulo de treinta grados con el plano vertical que contiene al eje de pista.
    - Zona de limitación de alturas: Es la superficie de terreno comprendida entre los planos e) y f) y dos planos verticales perpendiculares al eje de pista a distancia de cinco mil metros del punto de referencia y entre los planos e) y f), y otros dos verticales paralelos al eje de pista y situados a mil metros del punto de referencia.

- Superficie de limitación de alturas: Para la zona de seguridad será el Plano de Referencia. En el exterior de la zona de seguridad, dentro de los diedros formados por los planos e) y f) que contienen al eje de pista y su prolongación, la superficie de limitación de alturas estará formada por dos planos, que parten del punto de referencia y forman con el plano de este nombre una pendiente del dos por ciento. En los diedros, que no contienen al eje de pista ni su prolongación, la superficie de limitación de alturas estará formada por dos planos que contengan las intersecciones de las planos e) y f) con los planos inclinados anteriores.

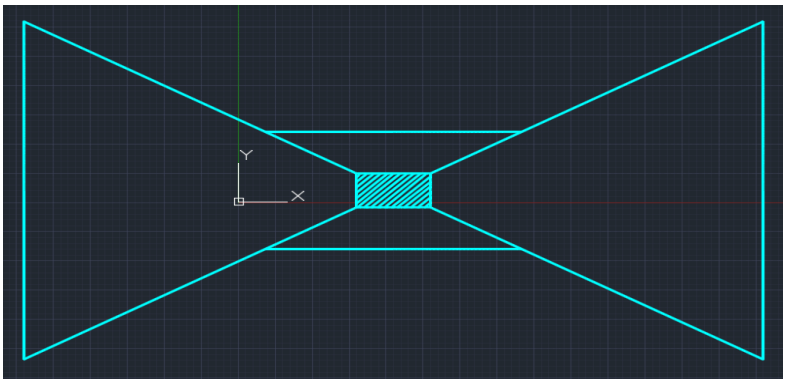

Figura 1.5 Servidumbres del ILS/LOC.

- Equipo de Planeo del sistema de aterrizaje instrumental (GP/ILS)
  - Zona de seguridad: Estará definida por dos planos verticales paralelos al eje de pista y distantes del punto de referencia de la instalación D, más D', más doscientos metros hacia la pista, y doscientos metros en sentido contrario (siendo D la distancia en metros del punto de referencia al eje de pista y D' la mitad de la anchura de la pista, en metros) y dos planos verticales, a) y b), perpendiculares a los anteriores y distantes del punto de referencia D'', más seiscientos metros hacia la cabecera de la pista y doscientos metros en sentido contrario (siendo D'' la distancia en metros del punto de referencia de la instalación al umbral de la pista).
  - Zona de limitación de alturas: Estará formada por la zona de seguridad y, además, por las porciones de terreno comprendidas entre dos planos verticales que pasen por el punto de referencia de la instalación y formen con el plano vertical que contiene al eje de pista ángulos de veinte grados y un plano paralelo al a) y a

una distancia D, más cinco mil metros del punto de referencia hacia la cabecera de la pista.

- Superficie de limitación de alturas: Estará definida por el plano de referencia hasta su intersección con el plano a) y, a partir de ella, por un plano de pendiente del 2 %.

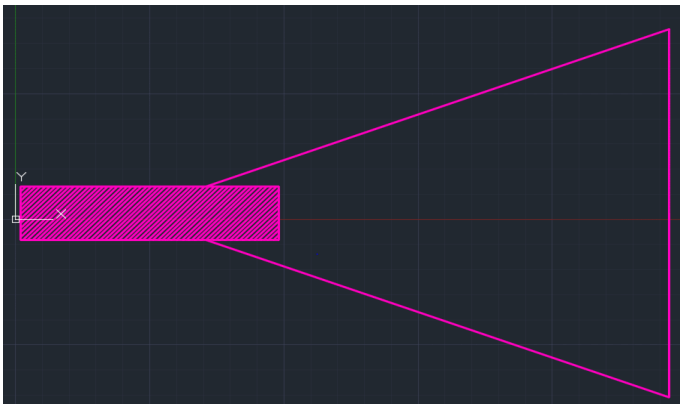

Figura 1.6 Servidumbres del ILS/GP.

1.3 Servidumbres aeronáuticas y seguridad aérea

Como ya se ha mencionado en anteriores apartados, el objetivo principal de la implementación de las servidumbres aeronáuticas es la de garantizar la seguridad y eficacia en las operaciones y maniobras, que tienen lugar en la infraestructura aeroportuaria y en sus alrededores.

Por lo cual, para que el funcionamiento de todas las instalaciones sea eficaz, es necesario que toda administración tanto de ámbito local, nacional o internacional, así como también las entidades particulares, respeten las regulaciones y restricciones que imponen las servidumbres. Estas normas son de obligado cumplimiento, su finalidad es garantizar el cumplimiento de los requerimientos necesarios para preservar la seguridad aérea y con ello, la vida de las personas.

En este apartado, se quiere insistir no sólo en la importancia de las servidumbres aeronáuticas, sino también en la transcendencia de que éstas hayan sido diseñadas de manera correcta. Debido a que un error en estos ámbitos, no solo conlleva importantes repercusiones económicas sino que también, puede derivar en la pérdida de vidas humanas. Para ello, se muestra documentación relativa a varios accidentes aéreos que han tenido lugar en los últimos años y entre las causas de los accidentes están el mal diseño de las SLO.

EL APARATO SE SALIÓ DE LA PISTA

Al menos 200 muertos al estrellarse un avión de TAM en un aeropuerto de Sao Paulo

- La pista estaba mojada y había sido recientemente reformada | Vea el vídeo
- El Ministerio de Exteriores no tiene constancia de que viajaran españoles en el avión
- Listado de pasajeros, tripulantes y desaparecidos

Actualizado miércoles 18/07/2007 15:10 (CET)

AGENCIAS

SAO PAULO.- Un avión de pasajeros de la aerolínea brasileña TAM con 186 personas a bordo se ha estrellado contra una gasolinera en el aeropuerto Congonhas de Sao Paulo. Por causas que aún se desconocen, el piloto no logró frenar a tiempo tras el aterrizaje, **colisionando contra un edificio de tres plantas y una gasolinera** cercanos a la pista.

La aeronave siniestrada es un Airbus A320 que había partido de Porto Alegre a las 17.16 horas (22.16 en la España peninsular) con destino a Sao Paulo. De los **186 ocupantes** del vuelo JJ3054, 162 eran pasajeros, 18 empleados de la empresa y seis tripulantes (dos comandantes y cuatro comisarios).

Portavoces de los bomberos ya han descartado la posibilidad de encontrar supervivientes entre los restos del edificio y del avión, que quedó totalmente calcinado.

El Ministerio de Exteriores no tiene constancia de que viajaran españoles en el avión.

Vídeo: Atlas

Figura 1.7 Detalle de la noticia sobre el accidente aéreo ocurrido en el aeropuerto de Sao Paulo, 2007. [4]

El primer accidente, en el que murieron 200 personas, tuvo lugar en el Aeropuerto de Congonhas (Sao Paulo, Brasil) en el año 2007, el aeropuerto está emplazado en un plano elevado, que debido al crecimiento de la ciudad está rodeado de grandes edificios. Una de las razones de las graves consecuencias del accidente fue la ubicación de los depósitos de combustible en la zona de transición.

A pesar de que gran parte de la culpa del accidente, la tiene las condiciones geográficas en la que está situado el aeropuerto (en el centro de la ciudad). Se quiere destacar la posibilidad de que varios edificios, aunque estén situados a un nivel de elevación inferior al aeropuerto, no respeten las servidumbres, ya que vulneran la línea de edificios de la superficie de transición. [4][5]

EN LA LOCALIDAD DE SANT QUIRZE DEL VALLÈS

Cuatro muertos en Barcelona al estrellarse una avioneta contra una grúa de la construcción sin balizar

EFE

SANT QUIRZE (BARCELONA).- Cuatro personas han muerto en Sant Quirze del Vallès (Barcelona) al estrellarse una avioneta contra una grúa de la construcción que no estaba balizada. Según los Bomberos de la Generalitat, los fallecidos eran **los ocupantes del aparato**, que era privado y de uso lúdico.

NOTICIAS RELACIONADAS

Los cuatro fallecidos son **Albert G.C.**, vecino de Barcelona y piloto de la aeronave, y los acompañantes **Antonio M.B.** y **Ferran C.R.**, ambos vecinos de Ripollet (Barcelona), y **Xavi M.M.**, vecino de Sitges.

La grúa estaba situada **en un edificio de cuatro plantas** que se encontraba **en obras**, lo que ha evitado una tragedia mayor.

El accidente ha tenido lugar hacia las 16.30 horas en la zona de Mas Duran, cerca de la estación de los ferrocarriles de la Generalitat, **en un área próxima a una gran superficie comercial de la firma francesa Alcampo** y donde se está edificando una promoción de nuevas viviendas.

Figura 1.8 Detalle de la noticia sobre el accidente aéreo ocurrido en el aeródromo de Sant Quirze del Vallés, 2005. [6]

2 años antes, en el 2005, en el Aeródromo de Sant Quirze del Vallés se instaló una grúa de grandes dimensiones en las proximidades del aeródromo, ocasionando un accidente, en el que murieron 4 personas. Debido a que la grúa vulneraba las servidumbres invadiendo el espacio aéreo, el piloto de la avioneta no pudo esquivar la grúa durante la fase de despegue. [6]

De tal manera, es de vital importancia que se diseñen las servidumbres de acuerdo con la normativa vigente, así como también asegurar que se van a respetar éstas. Manteniendo el espacio aéreo libre de obstáculos y poder así avalar la seguridad y eficiencia operacional.

Capítulo 2. Desarrollo de la aplicación

2.1 Descripción general de la aplicación

La aplicación desarrollada se ha llevado a cabo en el entorno de programación *AUTOCAD .NET*, en particular en el entorno *API.COM* en la cual es posible ejecutar y tomar el control de *AUTOCAD* de manera externa. [7]

Una de las principales características de la aplicación es que ésta ha sido diseñada para ejecutarse de manera independiente a *AUTOCAD*, por lo cual se trata de una aplicación *Out-of-process*. La otra alternativa son las aplicaciones que se ejecutan a la vez que el programa AUTOCAD (aplicación anfitriona) conocidas como aplicaciones *In-process*. [8]

Como se ha mencionado en la introducción, el principal objetivo del trabajo es el desarrollo de una aplicación cuyas principales funciones son:

- Determinar las servidumbres aeronáuticas de un aeropuerto a partir de la introducción de los principales parámetros de la pista del aeropuerto.
- Determinar si existe intersección entre las servidumbres y uno o varios posibles obstáculos introducidos por el usuario.
- Representar y dar al usuario la posibilidad de guardar en un archivo *.dwg* las servidumbres aeronáuticas.

Para hacer esto posible, es necesario procesar la información introducida y a partir de ahí, obtener el diseño de las servidumbres siguiendo la recomendación de la normativa (Anexo 14 en caso de las SLO, Decreto 584/1972 en caso de las servidumbres radioeléctricas). Una vez que ya se tengan las servidumbres determinadas, ya se podrán representar en el aplicativo *AUTOCAD* y también diagnosticar si existe algún obstáculo que esté vulnerando la seguridad aérea de la infraestructura aeroportuaria.

Otro punto importante a tener en cuenta, es que todas las distancias con las que el simulador trabaja están en metros. Por lo que es muy importante, que toda la información relativa a distancias que el usuario introduzca en el programa estén en éstas unidades.

2.2 Arquitectura de la aplicación

El simulador consta de 2 partes:

- Una interfaz principal a partir de la cual, el usuario puede introducir los datos necesarios para determinar las servidumbres, además de la información referente a los obstáculos. En este bloque de la aplicación, es donde se determinan si los obstáculos invaden el espacio aéreo definido por las servidumbres.
- Un segundo bloque que se ejecuta mediante el aplicativo *AUTOCAD*, donde el usuario puede ver representados tanto las servidumbres físicas como las radioeléctricas. Además, de poder guardarlas en un archivo *.dwg* y así poder consultarlo en un futuro sin la necesidad de ejecutar el programa otra vez.

2.3 Programas utilizados

Para el diseño y el desarrollo del simulador se ha utilizado *Microsoft Visual Studio 2012* y *AUTOCAD 2015*.

El diseño se ha realizado en *Microsoft Visual Studio 2012*. A partir del cual, se ha creado la interfaz, con el cual el usuario interactuará la mayor parte del tiempo, exceptuando únicamente cuando se quiera representar las servidumbres. Así pues, la interacción entre usuario y aplicación (introducción de datos, exposición de datos procesados) se realiza mediante *Forms*.

El uso de *AUTOCAD 2015,* se reduce a la representación de las servidumbres, la principal razón por la que se ha escogido *AUTOCAD 2015* es debido a que un importante porcentaje de planos son desarrollados actualmente en el formato de este programa.

2.4 Requisitos necesarios

Además de tener instalados los programas mencionados en el anterior apartado, para poder ejecutar el programa, es necesario descargarse y agregar una serie de librerías *.dll*.

Estas librerías permiten conectar la aplicación con el entorno *AUTOCAD*, de manera que podremos declarar nuevas variables y atributos en nuestra aplicación desarrollada en *Visual Studio*, que posteriormente se ejecutarán en *AUTOCAD.*

Los pasos que a seguir para descargar las librerías, se encuentran en el siguiente enlace [9].

Es importante que nos descarguemos las librerías correspondientes a la versión de *AUTOCAD,* que tengamos instalada en nuestro ordenador.

Una vez tengamos las librerías descargadas se tiene que referenciar en la aplicación de Visual Basic. En caso del *AUTOCAD 2015*, es necesario referenciar las siguientes librerías.

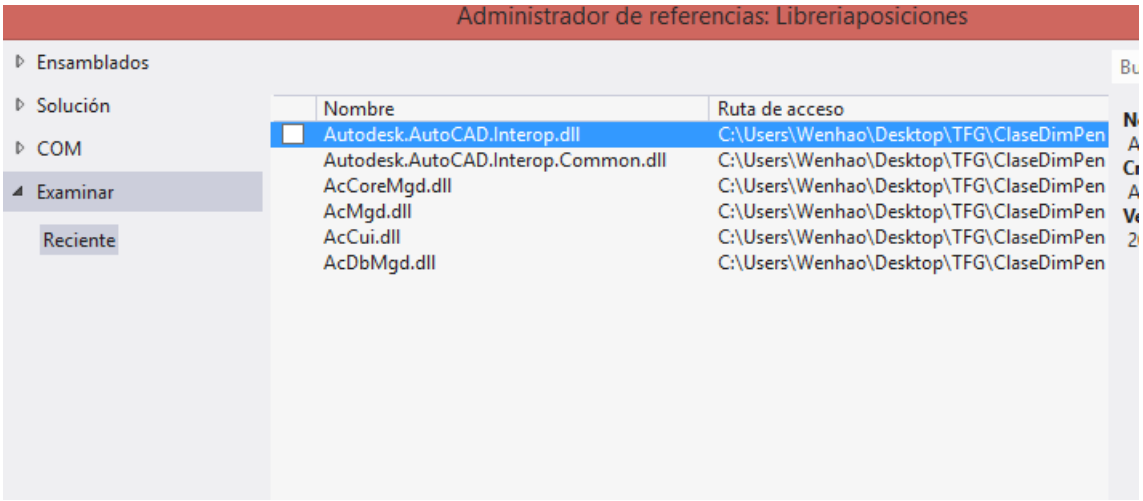

Figura 2.1 Librerías a referenciar si se trabaja con *AUTOCAD 2015*.

Finalmente, para acabar el proceso de referenciado no hay que olvidarse de aplicar las directivas *using*.

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Interop.Common;
using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.DatabaseServices;
```

Figura 2.2 Directivas *using* a aplicar.

2.5 Lenguaje de programación

Microsoft Visual Studio 2012, ofrece la posibilidad de programar en varios lenguajes como ahora Visual Basic, C++ o C# entre otros. De todos estos, se ha optado por el lenguaje C# debido a que es un lenguaje orientado a objetos y a pesar de que su sintaxis es muy expresiva, es un lenguaje sencillo y fácil de entender.

Otra de las razones por las cuales se ha escogido el lenguaje C# es su capacidad de interoperabilidad, que habilita los programas de C# a interactuar con otro software de Windows, como objetos COM o archivos DLL. [10]

Capítulo 3. Solución adoptada

Como se ha indicado con anterioridad, entre las principales funcionalidades del programa se halla la de introducir una serie de datos y a partir de éstos la aplicación mostrará al usuario un conjunto de resultados, que pueden ser de manera visual (representación de servidumbres) o como una sucesión de datos (información relativa a las intersecciones).

En este apartado, se explicarán las estrategias que el programa tomará durante los procesos para la determinación de intersecciones. Así como también, los procedimientos que se seguirán para definir los puntos necesarios para representar las servidumbres.

Finalmente, también se comentarán otros aspectos como la introducción, el procesamiento o la modificación de los diversos parámetros de entrada.

3.1 Determinación de las intersecciones

Los procesos encargados de fijar si alguno de los objetos intersectan con las servidumbres, son llevadas a cabo por los métodos definidos en la librería *Intersecciones*.

Estos métodos, que devuelven un booleano, están constituidos por un conjunto de condiciones y ecuaciones. En caso de que el objeto cumpla las condiciones, se devuelve un booleano con valor *true* significando de que existe una intersección, en caso contrario se devuelve un booleano con valor *false*.

Se considera que existe intersección cuando se cumplen las siguientes condiciones:

a. La profundidad o elevación *z* del objeto es igual o superior a la profundidad de la superficie en las coordenadas en las cuales está emplazado el objeto. Para determinar la *z* de la superficie (exceptuando el caso de la Superficie de Horizontal Interna y la zona de seguridad de las superficies radioeléctricas que son constantes) se aplica la siguiente expresión:

$$z_{sup} = \frac{P}{100} * \left(a_{pos} - a_0\right) + z_0 \qquad \textbf{(3.1)}$$

Donde:

- z_{sup} es el valor de la profundidad de la superficie en las coordenadas del obstáculo.
- P es la pendiente de la superficie.
- a_{pos} es el valor de x o la y del obstáculo,
- a_o es el valor de la x o la y en el inicio de la superficie.
- z_o es el valor de la z en el inicio de la superficie.

En caso de que a_{pos} sea una coordenada *x*, la variable a_o también lo será. Por lo cual, dependiendo de la superficie que se esté analizando se utilizará en la expresión *3.1 x* o *y,* más adelante se especificará superficie por superficie la coordenada utilizada.

b. El objeto está dentro de los límites dimensionales ($y_{máx}$-$y_{mín}$, $x_{máx}$-$x_{mín}$) que abarca la superficie.

Exceptuando la Superficie de Aproximación Interna, que es un plano rectangular inclinado, el resto de las superficies no tienen una composición geométrica tan simple, razón por la cual para analizar la presencia de intersecciones se descompone las superficies en formas geométricas más simples.

Si nos fijamos la siguiente *Figura 3.1*, se aprecia una Superficie de Aproximación interna (rectángulo naranja) y una Superficie de Aproximación, para una pista que está diseñada para vuelos visuales o de no precisión (polígono rojo).

El procedimiento de determinación de la profundidad es prácticamente el mismo en ambas superficies. Pero, para saber si el objeto está dentro de los límites es mucho más simple en la Superficie de Aproximación Interna, debido a que tiene forma rectangular. Por ello, para simplificar el procedimiento de la Superficie de Aproximación, se divide la superficie en 3 sectores (1 rectangular y 2 triangulares), en la *Figura 3.1* la partición se ha realizado mediante 2 líneas blancas.

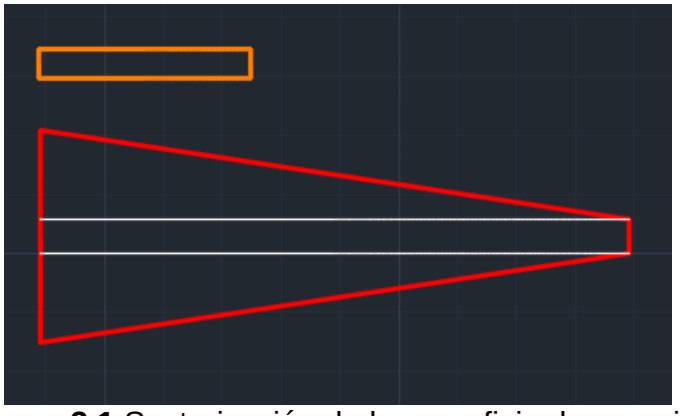

Figura 3.1 Sectorización de la superficie de aproximación y aproximación interna.

De tal forma, ya que las superficies presentan diferentes formas, se pueden descomponer al final en 4 formas geométricas:

- Cuarto de círculo.
- Cuarto de corona circular.
- Rectángulo.
- Triángulo.

Figura 3.2 Tipos de sectores.

En caso del sector rectangular, para determinar si un objeto está dentro de sus límites, simplemente hay que comprobar si las coordenadas del objeto están dentro del intervalo de valores de los límites dimensionales del sector.

No se puede decir lo mismo de los otros casos, ya que el valor de sus *x* e *y* varían a lo largo del contorno del sector. Por lo cual, para determinar si un objeto está dentro de estos sectores sería necesario calcular el valor de la *x* o la *y* (dependiendo del caso se calculará uno u otro, más adelante se especificará para cada caso) del contorno. Para ello, se utilizan las siguientes expresiones.

- Cuarto de círculo y cuarto de corona circular.

$$x_{sup} = \sqrt{R^2 - (y_{obj} - y_{pista})^2} + x_{pista} \qquad \textbf{(3.2)}$$

Donde:

- x_{sup}, es el valor de la x en el arco (obtendremos dos valores uno para el sector superior y otro para el sector inferior. Debido a que normalmente, las superficies que tienen un sector de con esta forma estarán formados por 3 sectores más con esta forma, pero con otra disposición es decir con una orientación del borde circular diferente).
- R, es el radio de la Cónica o de la Horizontal interna dependiendo del límite que deseemos calcular.
- y_{obj}, es el valor de la coordenada y del objeto.

- y_{pista}, es el valor de la coordenada y de la pista.

En el caso de la corona circular, es necesario calcular 2 x_{sup} debido a que se tienen 2 bordes circulares, el único parámetro que varía para determinar las 2 x_{sup} es el radio.

Cuando se esté trabajando con superficies radioeléctricas se utilizarán las coordenadas de la instalación radioeléctrica y no los de la pista.

- Triángulo

$$y_{sup} = |x_{obj} - x_x| * \frac{D}{100} + y_y \quad \textbf{(3.3)}$$

Donde:

- y_{sup}, es el valor de la y a partir de la coordenada x del objeto.
- x_{obj}, es el valor de la x del objeto.
- x_x es el valor de la x al inicio de la divergencia (el punto donde la altura empieza a variar).
- y_y, es el valor de la y al inicio de la divergencia (el punto donde la altura empieza a variar).
- D, es el valor de la divergencia de la superficie.

Se aplica el valor absoluto debido a que en algunos casos x_{obj} es inferior a x_x pero y_{sup} aumenta como en el caso de la *Figura 3.3*.

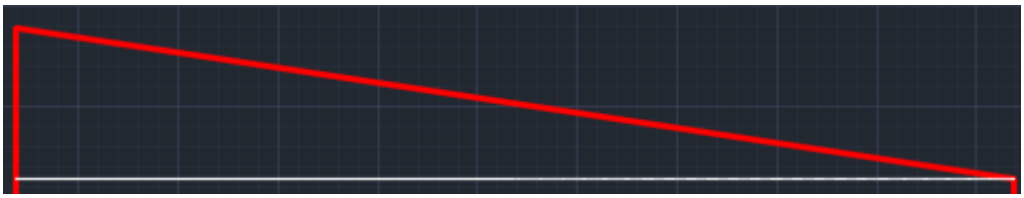

Figura 3.3 Sector triangular donde la y crece a medida que la x decrece.

En otros casos, como en el que se muestra en la *Figura 3.4*, el valor de la divergencia es negativo debido a que el valor de la y_{sup} disminuye.

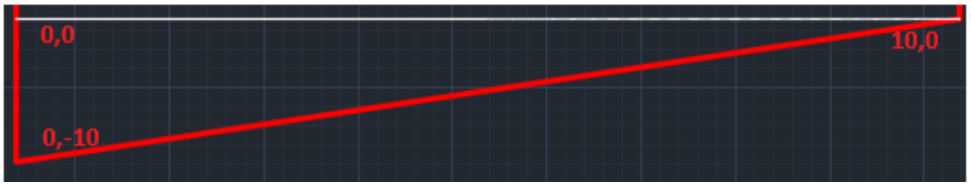

Figura 3.4 Sector triangular donde la y decrece a medida que la x decrece.

Con la aplicación de estas fórmulas, se fija el valor de la *x* o la *y* (sabiendo que una de ellas está dentro del sector) para determinar si la otra también está dentro del sector.

Por ejemplo, para el sector triangular de la anterior figura, sabiendo los límites, tenemos dos objeto con coordenadas 5,7 y 5,3 se sabe que la *x* de ambos objetos están dentro del intervalo de valores de la *x* del sector 0-10. A continuación, calcularemos cuánto vale el valor de *y* para *x* igual a 5. Entonces aplicando la *expresión 3.3*:

En este caso:

- D (divergencia) es igual a -100 negativo debido a que disminuye.
- x_{obj} es 5.
- x_x es 0.
- y_y es 10.

$$y_{sup} = |5 - 10| * \frac{-100}{100} + 10 = 5$$

Se obtiene que el valor es 5, esto quiere decir que el valor de la hipotenusa del triángulo vale 5, por lo cual, todos los valores que tengan un valor igual o inferior a 5 e igual o superior a 0 (límite inferior/cateto horizontal del triángulo) cumplirá la condición de que está dentro del sector triangular para una *x* de objeto igual a 5.

De tal forma, el primer objeto (*y*=7) no cumpliría la condición de estar dentro del sector y sería descartado, el segundo objeto (*y*=3) sí que cumple con la condición y se comprobaría si su profundidad *z* es mayor o igual a la *z* de la superficie en las coordenadas del objeto; en caso afirmativo existiría intersección.

Entoces, para determinar si un objeto vulnera una superficie:

- Primero, se comprueba si el objeto está dentro de los límites superficiales.

- Seguidamente, se identifica en que sector se halla el objeto.
- Finalmente, se comprueba si la elevación *z* es igual o superior a la *z* de la superficie en la posición del objeto.

3.2 Representación de las servidumbres

A partir de los límites dimensionales y un conjunto de distancias, que indican cuando tiene lugar un punto de inflexión en las superficies, obtenidas en la librería *Intersecciones*.

Se determinan las coordenadas de los puntos base que conforman el contorno y las curvas de nivel de las diferentes superficies, con los métodos desarrollados en la librería *Dibuclass.* Estos puntos son guardados en unas matrices y posteriormente se envían en forma de comandos a *AUTOCAD*.

Además de los datos mencionados anteriormente, también se hace uso de los parámetros de la pista, la franja y las características dimensionales de las superficies para definir las coordenadas de los bordes y las curvas de nivel.

Para facilitar la distinción entre los bordes y las curvas se representan los bordes con un grosos de línea más fino. Por otro lado, las superficies son dibujadas en diferente color de manera que sea más sencilla su identificación.

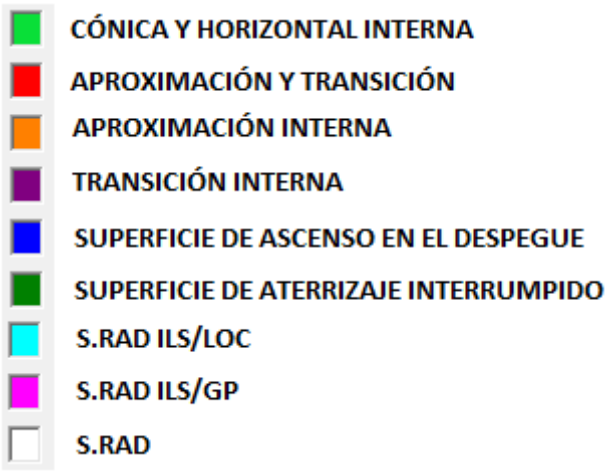

Figura 3.5 Paleta de colores con las que se representaran las servidumbres.

3.3 Procedimiento de análisis y obtención de parámetros

A continuación, se explicarán para cada superficie la manera a partir de la cual se determinarán las intersecciones y las matrices con las coordenadas para representar las superficies.

3.3.1 Horizontal interna y Cónica

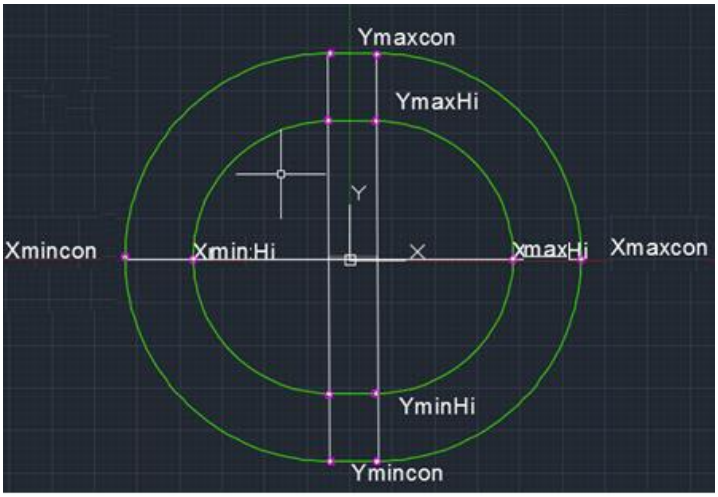

Figura 3.6 Límites superficiales de la Horizontal interna y la Cónica.

En la anterior figura, se puede apreciar en color magenta los puntos base que conforman el contorno de las superficies Cónica y Horizontal interna, las coordenadas de estos puntos son conocidos ya que están formados por los límites superiores e inferiores de las superficies.

Estos límites se han determinado a partir de las coordenadas del centro de la pista, de tal forma que:

$$X_{maxcon} = x_{pista} + \frac{l}{2} + (R\ + (\frac{H*100}{P})) \quad \textbf{(3.4)}$$

$$X_{mincon} = x_{pista} - \frac{l}{2} - (R\ + (\frac{H*100}{P})) \quad \textbf{(3.5)}$$

$$Y_{maxcon} = y_{pista} + (R\ + (\frac{H*100}{P})) \quad \textbf{(3.6)}$$

$$Y_{mincon} = y_{pista} - (R\ + (\frac{H*100}{P})) \quad \textbf{(3.7)}$$

$$X_{maxHi} = x_{pista} + \frac{l}{2} + R \quad \textbf{(3.8)}$$

$$X_{minHi} = x_{pista} - \frac{l}{2} - R \quad \textbf{(3.9)}$$

$$Y_{maxHi} = y_{pista} + R \quad \textbf{(3.10)}$$

$$Y_{minHi} = y_{pista} - R \quad \textbf{(3.11)}$$

Donde:

- l, es la longitud de la pista.
- R, es el radio de la Horizontal interna.
- H, es la profundidad *z* de la cónica por encima de la horizontal interna.

Por otro lado, la altura *y* de los puntos localizados a la altura del centro de los arcos es la altura *y* de la pista.

De tal manera, a partir de los valores de estos límites se obtienen los puntos que conforman las curvas de nivel, las coordenadas de las curvas de nivel se calculan a partir de las mismas expresiones utilizadas para determinar los límites de la cónica, solo que en lugar de utilizar *H* se utilizan incrementos de 5m desde la profundidad de la horizontal interna hasta llegar a *H*. El método que se encarga de determinar las curvas de nivel de la Superficie Cónica es *Getcon*.

Para determinar si existe intersección, en primer lugar se comprueba si el objeto supera los 45m de profundidad por encima del punto de referencia de la pista, si no es el caso el objeto nunca corta con estas servidumbres. Seguidamente, se determina si el objeto está dentro de los límites máximos y mínimos. En caso que sea afirmativo, se identifica en cuál de los sectores se encuentra, por lo que es necesario analizar si la altura *y* del objeto está por encima o por debajo de la altura *y* de la pista.

En caso de que esté dentro de uno de los sectores de la cónica, es necesario determinar con las coordenadas del objeto que profundidad *z* (a partir de la *expresión 3.1*) tiene la superficie ya que la profundidad de ésta va aumentando a medida que se aleja del borde interior de la cónica, en caso de que la profundidad del objeto sea mayor que la profundidad hallada existe intersección. Por otro lado, en la horizontal interna con que el objeto esté 45m o más por encima del plano de referencia habrá intersección.

En los sectores centrales (rectangulares), para comprobar si el objeto está dentro de los límites del sector simplemente hay que determinar si las coordenadas del objeto están dentro del intervalo de los límites del sector. En el resto de los sectores (cuarto de círculo o cuarto de corona), es necesario determinar el valor de la *x* del límite (arco) a partir de la *y* del objeto para saber si la *x* del objeto está dentro de esos límites (en caso de la Cónica, se comprueba también que el objeto esté por encima del arco formado por la Horizontal interna), este valor de *x* es calculado mediante la expresión *3.2*.

Se utiliza el valor menor de la *x* obtenida en la expresión *3.2* cuando el objeto se halle por debajo de la altura de la pista, mientras que la *x* de mayor valor se usa cuando el objeto se halle por encima de la altura de la pista. Finalmente, se aplican las siguientes condiciones para confirmar que el objeto se halla dentro del sector:

- Para los sectores de la horizontal interna (cuarto de círculo).

- En caso de que la x_{obj} sea mayor que la x_{pista}.

$$x_{obj} \leq x_{sup} \quad \textbf{(3.12)}$$

- En caso de que la x_{obj} sea menor que la x_{pista}.

$$x_{obj} \geq x_{sup} \quad \textbf{(3.13)}$$

- Para los sectores de la cónica (cuarto de corona circular).
  - En caso de que la x_{obj} sea mayor que la x_{pista}.

$$x_{supHi} \leq x_{obj} \leq x_{supcon} \quad \textbf{(3.14)}$$

  - En caso de que la x_{obj} sea menor que la x_{pista}.

$$x_{supHi} \geq x_{obj} \geq x_{sup} \quad \textbf{(3.15)}$$

Los métodos utilizados para la determinación de intersecciones para la Superficie Cónica y la Superficie Horizontal interna son *inthoint* e *intconi*, respectivamente.

3.3.2 Aproximación y Transición

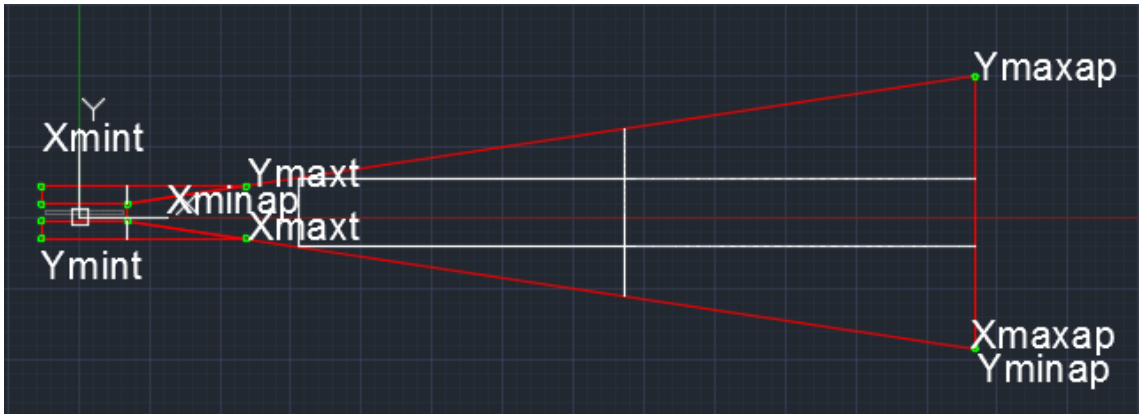

Figura 3.7 Límites superficiales de la Aproximación y la Transición.

Al igual que en el anterior caso, los puntos que conforman el contorno de ambas superficies están formadas por los límites superficiales.

A diferencia del resto de las superficies, las superficies de transición (transición y transición interna) están formadas por dos bloques, uno en cada lateral de la pista justo después de la franja.

Además de la superficie de transición, en la *Figura 3.7* se muestra también, la superficie de aproximación para una pista diseñada para aproximaciones de precisión, por lo cual la superficie de aproximación tiene dos secciones más, la sección secundaria y la sección horizontal.

Los límites han sido determinados a partir de las coordenadas del centro de la pista. Un factor que influye en el valor de los límites, es el sentido de aterrizaje de la pista ya que dependiendo del sentido, la orientación de las superficies no es la misma y consecuentemente las coordenadas de los límites en la *x* tampoco.

Por otro lado, se determinan unas coordenadas auxiliares que sirven para determinar los puntos donde empiezan/acaban determinados sectores.

- Superficie de transición.

$$Y_{maxt} = y_{pista} + (fancho + (45 * \frac{100}{P}) \quad \textbf{(3.16)}$$

$$Y_{mint} = y_{pista} - (fancho + (45 * \frac{100}{P}) \quad \textbf{(3.17)}$$

- Aterrizaje producido desde la izquierda.

$$X_{mint} = \left(\frac{100}{-D}\right) * \left(Y_{maxt} - \left(y_{pista} + \frac{lbointap}{2}\right)\right) + (x_{pista} - \frac{l}{2} - duap + posub) \quad \textbf{(3.18)}$$

$$X_{maxt} = (x_{pista} + \frac{l}{2} + flong) \quad \textbf{(3.19)}$$

$$x_1 = x_{min} \quad \textbf{(3.20)}$$

$$x_2 = x_{pista} - \frac{l}{2} - duap + posub \quad \textbf{(3.21)}$$

- Aterrizaje producido desde la derecha.

$$X_{maxt} = \left(\frac{100}{D}\right) * \left(Y_{maxt} - \left(y_{pista} + \frac{lbointap}{2}\right)\right) + (x_{pista} + \frac{l}{2} + duap - posub) \quad \textbf{(3.21)}$$

$$X_{mint} = (x_{pista} - \frac{l}{2} - flong) \quad \textbf{(3.22)}$$

$$x_1 = x_{max} \quad \textbf{(3.23)}$$

$$x_2 = x_{pista} + \frac{l}{2} + duap - posub \quad \textbf{(3.24)}$$

Donde:

- f_{ancho}, es el valor del ancho de la franja a un lado del eje de la pista.
- P, es la pendiente de la Superficie de transición.
- lbointap, es la longitud del borde interior de la superficie de aproximación.
- duap, es la distancia del borde interior de la Superficie de Aproximación hasta el umbral.
- posub, es la distancia a la cual está desplazada el umbral.
- x_1, es la coordenada x de cuando empiezan las secciones triangulares.
- x_2, es la coordenada x de cuando acaban las secciones triangulares.

- Superficie de Aproximación. Los límites de esta superficie son diferentes dependiendo del diseño de la pista, para pistas de vuelo visual (cualquier longitud) y pistas de aproximación de no precisión con longitud inferior a 1200m, la Superficie de Aproximación tiene únicamente una sección, para el resto de diseños la superficie tiene tres secciones con diferentes pendientes.

    - Pistas con solo 1 sección.

$$Y_{maxap} = y_{pista} + \frac{D}{100} * pseclong + \frac{lbointap}{2} \quad \textbf{(3.24)}$$

$$Y_{minap} = y_{pista} - \frac{D}{100} * pseclong - \frac{lbointap}{2} \quad \textbf{(3.25)}$$

        - Aterrizaje por la izquierda.

$$X_{maxap} = x_{pista} - \frac{l}{2} - duap + posub \quad \textbf{(3.26)}$$

$$X_{minap} = X_{maxap} - pseclong \quad \textbf{(3.27)}$$

$$XX = X_{maxap} \quad \textbf{(3.28)}$$

        - Aterrizaje por la derecha.

$$X_{mim} = x_{pista} + \frac{l}{2} + duap - posub \quad \textbf{(3.29)}$$

$$X_{max} = X_{min} + pseclong \quad \textbf{(3.30)}$$

$$XX = X_{min} \quad \textbf{(3.31)}$$

    - Pistas con 3 secciones. Los límites son determinados a partir de los límites de la primera sección, que se calculan de la misma manera que en el caso de 1 sola sección.

$$Y_{max2ap} = Y_{maxap} + \frac{sseclong+shorlong}{100} * D \quad \textbf{(3.32)}$$

$$Y_{min2ap} = Y_{minap} - \frac{sseclong+shorlong}{100} * D \quad \textbf{(3.33)}$$

        - Aterrizaje por la izquierda.

$$X_{max2ap} = X_{min} \quad \textbf{(3.34)}$$

$$X_{min2ap} = X_{max2ap} - sseclong - shorlong \quad \textbf{(3.35)}$$

$$x_1 = x_{min2ap} + shorlong \quad \textbf{(3.36)}$$

$$x_2 = x_{max2ap} \quad \textbf{(3.37)}$$

$$xx_2 = x_1 \quad \textbf{(3.38)}$$

$$xx = X_{maxap2} \quad \textbf{(3.39)}$$

- Aterrizaje por la derecha.

$$X_{min2ap} = X_{max} \quad \textbf{(3.40)}$$

$$X_{max2ap} = X_{min2ap} + sseclong + shorlong \quad \textbf{(3.41)}$$

$$x_2 = x_{max2ap} - shorlong \quad \textbf{(3.42)}$$

$$x_1 = x_{min2ap} \quad \textbf{(3.43)}$$

$$xx_2 = x_2 \quad \textbf{(3.44)}$$

$$xx = X_{min2ap} \quad \textbf{(3.45)}$$

Donde:

- D, es la divergencia de la Superficie de Aproximación.
- lbointap, es la longitud del borde interior de la superficie de aproximación.
- duap, es la distancia del borde interior de la Superficie de Aproximación hasta el umbral.
- posub, es la distancia a la cual está desplazada el umbral.
- pseclong, es la longitud de la primera sección.
- sseclong, es la longitud de la segunda sección.
- hseclong, es la longitud de la sección horizontal.
- xx_1, es el inicio de los sectores triangulares de la sección horizontal.
- xx_2, es el inicio de los sectores triangulares de la segunda sección.
- x_1 y x_2, es el intervalo de la segunda sección.

Una vez calculados los límites, se pueden obtener las curvas de nivel de las superficies.

En el caso de la Superficie de Transición se emplean las ecuaciones para determinar Y_{maxt} e Y_{mint} ,pero en lugar de 45m se utiliza incrementos de 5m profundidad que parte desde el valor de la profundidad de la pista hasta los

45m. Para determinar los valores de *x*, hay que tener en cuenta la divergencia de la Superficie de Aproximación, por el lado en el que se produce el aterrizaje, en el otro lado la *x* es constante.

Por otro lado, en el proceso de determinación de curvas de nivel de la Superficie de Aproximación hay que tener en cuenta que la pendiente es diferente dependiendo de la sección, además en la sección horizontal no hay curvas de nivel ya que es una sección totalmente plana. Las coordenadas de las *x*, se determinan a partir de incrementos de profundidad de 5m teniendo en cuenta la pendiente de la sección. Finalmente, las *y* se calculan sustituyendo la *pseclong* o *sseclong* + *hseclong* en las ecuaciones dependiendo de la sección en la cual se halle el objeto, por los valores de *x* una vez aplicado los incrementos de *z*.

Los métodos encargados de determinar las matrices con los bordes y las curvas de nivel, de las Superficie de Transición y Superficie de Aproximación son *GetTR* y *GetAP* de la librería *Dibuclass* respectivamente.

Para determinar las intersecciones, primero se tiene que calcular el valor de la profundidad en la superficie con las coordenadas del objeto a partir de la expresión *3.1*. Un factor a tener en cuenta, es que la pendiente de las diferentes secciones de la superficies de aproximación no son iguales, de ahí la razón por la que se han separado en diferentes sectores.

En los sectores centrales, de la Superficie de aproximación y exteriores de la Superficie de transición (rectangulares), para que haya intersección el objeto debe superar las profundidades mencionadas y estar dentro de los límites del sector.

En el resto de los sectores (triangulares), es necesario determinar el valor de la altura y de los límites a partir de las coordenadas del objeto para ver si la *y* del objeto sobrepasa la *y* de la superficie. Este valor de *y* se calcula a partir de la expresión *3.3*.

- Si la y_{obj} es mayor que la y_{pista}.
    - En la Superficie de Transición.

$$y_{obj} \geq y_{sup} \quad \textbf{(3.46)}$$

    - En la Superficie de Aproximación.

$$y_{obj} \leq y_{sup} \quad \textbf{(3.47)}$$

- Si la y_{obj} es menor que la y_{pista}.
    - En la Superficie de Transición.

- $y_{obj} \leq y_{sup}$ **(3.48)**

  - En la Superficie de Aproximación.

- $y_{obj} \geq y_{sup}$ **(3.49)**

Los métodos encargados para la determinación de las intersecciones de la Superficie de Aproximación y de la Superficie de Transición son i*ntaprox* e *inttrans* de la librería I*ntersecciones*, respectivamente.

3.3.3 Superficie de aterrizaje interrumpido y Superficie de ascenso en el despegue

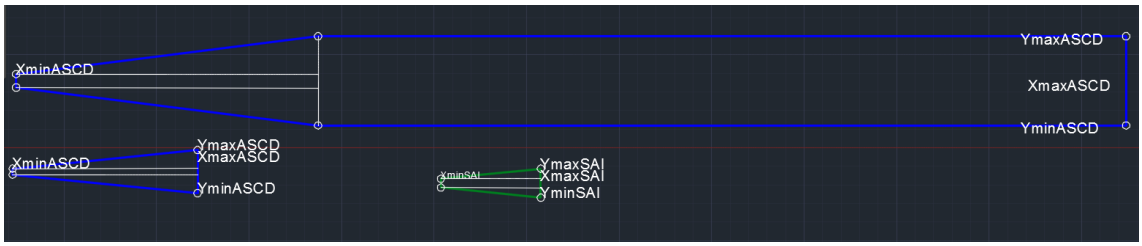

Figura 3.8 Límites superficiales de la Superficie de aterrizaje interrumpido y la Superficie de ascenso en el despegue.

A diferencia del caso de la Superficie de Ascenso en el Despegue para pistas con una longitud superior a 1200 m (polígono azul superior de la *Figura 3.8*), el cual tiene una sección rectangular. La determinación de intersecciones y la obtención de la matriz con las coordenadas de los puntos para la representación para ambas superficies, se realizan prácticamente de la misma manera que en el caso de la primera sección de la Superficie de Aproximación. La principal diferencia entre ambos procesos es la aplicación de las características dimensionales (pendiente, divergencia o longitudes) para la obtención de los puntos.

- Superficie de Aterrizaje Interrumpido (SAI).

$$Y_{maxSAI} = y_{pista} + \frac{lbsai}{2} + \frac{45*100}{P} \quad \textbf{(3.50)}$$

$$Y_{minSAI} = y_{pista} - \frac{lbsai}{2} - \frac{45*100}{P} \quad \textbf{(3.51)}$$

- Aterrizaje por la izquierda.

$$X_{minSAI} = x_{pista} + possai \quad \textbf{(3.52)}$$

$$X_{maxSAI} = X_{minSAI} + \frac{45*100}{P} \quad \textbf{(3.53)}$$

$$XX = X_{minSAI} \quad \textbf{(3.54)}$$

- Aterrizaje por la derecha.

$$X_{maxSAI} = x_{pista} - possai \quad \textbf{(3.355)}$$

$$X_{minSAI} = X_{maxSAI} - \frac{45*100}{P} \quad \textbf{(3.56)}$$

$$XX = X_{maxSAI} \quad \textbf{(3.57)}$$

Donde:

- lbsai, es la longitud del borde interior de la SAI.
- P, es la pendiente de la SAI.
- possai, es la distancia a la cual está situado la SAI después del umbral.
- XX, es la coordenada x donde empieza los sectores triangulares.

- Superficie de Ascenso en el despegue (ASCD). Antes de definir los límites es importante determinar si la longitud de la pista es igual o superior a 1200m. Si es el caso, el valor de algunas variables son diferentes.

$$Y_{maxASCD} = y_{pista} + \frac{anchASCD}{2} \quad \textbf{(3.58)}$$

$$Y_{minASCD} = y_{pista} - \frac{anchASCD}{2} \quad \textbf{(3.59)}$$

- Despegue por la derecha.

$$X_{minASCD} = x_{pista} + dpASCD + \frac{l}{2} \quad \textbf{(3.60)}$$

$$XX_1 = X_{minASCD} \quad \textbf{(3.61)}$$

$$X_{maxASCD} = X_{minASCD} + LASCD \quad \textbf{(3.62)}$$

- Si la longitud es igual o superior a los 1200m.

$$XX_2 = X_{minASCD} + \frac{(anchASCD - lbASCD) * 100}{2 * D} \quad \textbf{(3.63)}$$

- Si la longitud es inferior a 1200m.

$$XX_2 = X_{maxASCD} \quad \textbf{(3.64)}$$

- Despegue por la izquierda.

$$X_{maxASCD} = x_{pista} - dpASCD - \frac{l}{2} \quad \textbf{(3.65)}$$

$$XX_1 = X_{maxASCD} \quad \textbf{(3.66)}$$

$$X_{minASCD} = X_{maxASCD} - LASCD \quad \textbf{(3.67)}$$

- Si la longitud es igual o superior a los 1200m.

$$XX_2 = X_{maxASCD} - \frac{(anchASCD - lbASCD) * 100}{2 * D} \quad \textbf{(3.68)}$$

- Si la longitud es inferior a 1200m.

$$XX_2 = X_{minASCD} \quad \textbf{(3.69)}$$

Donde:

- anchASCD, es la anchura del borde exterior de la ASCD.
- dpASCD, es la distancia hasta el extremo de pista de la ASCD.
- lASCD, es la longitud de la ASCD.
- lbASCD, es la longitud del borde interior de la ASCD.
- D, es la divergencia de la ASCD.

Como ya se ha comentado, el proceso para obtener las curvas de nivel es bastante similar al caso de la Superficie de Aproximación. Para obtener las coordenadas de la *x*, en el caso de la SAI y en el tramo triangular de la ASCD se sustituiría 45m por incrementos de profundidad de 5m hasta llegar a los 45m, en cuanto a los valores de *y* se obtienen a partir del producto de las *x* con la respectiva divergencia. Para las pistas que tienen una longitud igual o superior a 1200m, el valor de las *x* se determinan de la misma manera, en cambio las y permanecen constantes por lo cual tienen el mismo valor que el de las *y* de los límites superficiales. Los métodos encargados de obtener estas coordenadas son *GetSAI* y *GetASCD*.

De la misma manera, una vez que se haya identificado en que sector se encuentra el objeto. Los métodos que se encargan de determinar la existencia de intersecciones son *intsai* e *intasc* y el procedimiento que siguen para determinar si un objeto se puede considerar obstáculo es el mismo que se ha aplicado para los sectores triangulares *y* rectangulares de las superficies explicadas en los anteriores apartados.

3.3.4 Aproximación interna y Transición interna

Figura 3.9 Límites superficiales de la Aproximación interna y la Transición interna.

En la *Figura 3.9* se pueden apreciar las Superficie Aproximación interna (lila) y la Superficie de Aproximación interna (naranja). Además de indicar los límites superficiales de las dos superficies, se han nombrado los sectores triangulares de la Superficie de transición interna como 1 (sectores de la derecha) y 2 (sectores de la izquierda). Más adelante, se explicará la razón por la cual se realiza esta distinción.

Los límites superficiales vienen definidos de la siguiente manera:

- Aproximación Interna.

$$Y_{maxai} = y_{pista} + \frac{anchai}{2} \quad \textbf{(3.70)}$$

$$Y_{minai} = y_{pista} - \frac{anchai}{2} \quad \textbf{(3.71)}$$

- Aterrizaje por la izquierda

$$X_{maxai} = x_{pista} - \frac{l}{2} - duai + posub \quad \textbf{(3.72)}$$

$$X_{minai} = X_{maxai} - lai \quad \textbf{(3.73)}$$

$$XX = X_{maxai} \quad \textbf{(3.74)}$$

- Aterrizaje por la derecha

$$X_{minai} = x_{pista} + \frac{l}{2} + duai - posub \quad \textbf{(3.75)}$$

$$X_{maxai} = X_{minai} + lai \quad \textbf{(3.76)}$$

$$XX = X_{minai} \quad \textbf{(3.77)}$$

- Transición Interna.

$$Y_{maxti} = y_{pista} + \frac{anchai}{2} + 45 * \frac{100}{Pti} \quad \textbf{(3.78)}$$

$$Y_{minti} = y_{pista} - \frac{anchai}{2} - 45 * \frac{100}{Pti} \quad \textbf{(3.79)}$$

- Aterrizaje por la izquierda.

$$XX_1 = x_{pista} - \frac{l}{2} - duai + posub \quad \textbf{(3.80)}$$

$$X_{minti} = XX_1 - lai \quad \textbf{(3.81)}$$

$$XX_2 = x_{pista} + possai \quad \textbf{(3.82)}$$

$$X_{maxti} = XX_2 + 45 * \frac{100}{Psai} \quad \textbf{(3.83)}$$

- Aterrizaje por la derecha.

$$XX_1 = x_{pista} + \frac{l}{2} + duai + posub \quad \textbf{(3.84)}$$

$$X_{maxti} = XX_1 + lai \quad \textbf{(3.85)}$$

$$XX_2 = x_{pista} - possai \quad \textbf{(3.86)}$$

$$X_{minti} = XX_2 - 45 * \frac{100}{Psai} \quad \textbf{(3.87)}$$

Donde:

- anchai, es la anchura del borde de la Aproximación interna.
- duai, es la distancia hasta el umbral de la Aproximación interna.
- lai, es la longitud de la Aproximación interna.
- posub, es la distancia a la cual está desplazado el umbral.
- Psai, es la pendiente de la SAI.
- Pti, es la pendiente de la Transición Interna.
- l, es la longitud de la pista.

Una vez se haya comprobado que el objeto supera la profundidad de la superficie (z_{sub}) calculado a partir de la expresión *3.1*. Se identifica en que sector se encuentra.

La Superficie de Aproximación Interna está formada únicamente por un sector rectangular, mientras que la Superficie de Transición Interna esta compuesto por dos sectores rectangulares y cuatro sectores triangulares.

En los sectores rectangulares, con que las coordenadas del objeto estén dentro de los límites queda demostrada la existencia de una intersección. Siempre que, la elevación del objeto sea igual o superior a la elevación de la superficie en ese punto.

Por otro lado, en la Superficie de Transición Interna es importante determinar si el objeto se encuentra por encima o por debajo de la altura *y* de la pista, ya que dependiendo de esto el valor de la divergencia que se aplica en la expresión *3.3,* es positivo para los sectores triangulares de por encima de la pista y negativo para el resto de los sectores triangulares. Además, dependiendo del

sector triangular donde esté localizado el objeto, la condición para demostrar que está dentro es diferente.

- Para los sectores triangulares 1(cuando $y_{obj}>=y_{pista}$) y 2 (cuando $y_{obj}<=y_{pista}$).

$$y_{obj} \geq y_{sup} \quad \textbf{(3.88)}$$

- Para los sectores triangulares 1(cuando $y_{obj}<=y_{pista}$) y 2 (cuando $y_{obj}>=y_{pista}$).

$$y_{obj} \leq y_{sup} \quad \textbf{(3.89)}$$

Los métodos desarrollados para la determinación de intersecciones utilizados para la Superficie de Transición Interna y la Superficie de Aproximación Interna son *inttrans* e *intaprox*.

Por otro lado, los métodos utilizados para la obtención de las matrices, con las coordenadas de los puntos para poder representar las superficies son *GetTrI* y *GetApI*.

El procedimiento seguido para obtener los puntos en la Superficie de Transición Interna es el mismo que el de la Transición Interna, la principal diferencia es que la Transición Interna las *x* de ambos lados van variando lo que hay que incrementar (sector triangular 2), y reducir (sector triangular 1) el valor de *x* a medida que incrementamos el valor de la *z* (profundidad) para definir las curvas de nivel. Para la Superficie de Transición Interna, se sigue el mismo procedimiento que en el caso del sector rectangular de la Superficie de Ascenso en el Despegue.

3.3.5 Servidumbres radioeléctricas

A diferencia de las servidumbres físicas, las servidumbres radioeléctricas poseen una zona de seguridad en la cual no se puede construir ni situar ningún objeto sin previo autorización de Ministerio del Aire. Por lo cual, siempre que se descubra un objeto en la zona de seguridad se considerará como un obstáculo, a no ser que su presencia esté justificada por las autoridades pertinentes.

Por otro lado, las servidumbres radioeléctricas poseen una zona de limitación de alturas, en la cual a medida que la zona se va extendiendo la profundidad es mayor debido a la pendiente ascendente de la servidumbre. Así que, para determinar la intersección es necesario calcular el valor de la profundidad de la superficie en el punto donde se localiza el objeto como en el resto de las servidumbres.

Esta peculiaridad, también afecta a la representación de las servidumbres ya que hay que representar tanto la zona de seguridad (zs) como la zona de limitación de alturas (zla).

Otra diferencia de las servidumbres radioeléctricas de las servidumbres físicas es, que el punto de referencia a partir del cual se expanden las superficies es el lugar donde esté emplazado la instalación radioeléctrica (IR) y no el centro de la pista como es el caso de las servidumbres físicas. De tal forma que:

$$Y_{maxSr} = y_{IR} + R_{zla} \quad \textbf{(3.90)}$$

$$Y_{minSr} = y_{IR} - R_{zla} \quad \textbf{(3.91)}$$

$$X_{maxSr} = x_{IR} + R_{zla} \quad \textbf{(3.92)}$$

$$X_{minSr} = x_{IR} - R_{zla} \quad \textbf{(3.93)}$$

$$Y_{maxzs} = y_{IR} + R_{zs} \quad \textbf{(3.94)}$$

$$Y_{minzs} = y_{IR} - R_{zs} \quad \textbf{(3.95)}$$

$$X_{maxzs} = x_{IR} + R_{zs} \quad \textbf{(3.96)}$$

$$X_{minzs} = x_{IR} - R_{zs} \quad \textbf{(3.97)}$$

Donde los radios son los de la zona de seguridad y la zona de limitación de alturas. Una vez se haya determinado el sector en el cual está situado el objeto se utilizaría la expresión *3.2*. Por lo tanto, para que se cumpla la condición de que el objeto está dentro de la superficie:

- Para los sectores de la zona de seguridad (cuarto de círculo).
  - En caso de que la x_{obj} sea mayor que la x_{IR}.

$$x_{obj} \leq x_{sup} \quad \textbf{(3.98)}$$

  - En caso de que la x_{obj} sea menor que la x_{IR}.

$$x_{obj} \geq x_{sup} \quad \textbf{(3.99)}$$

- Para los sectores de la zona de limitación de alturas (cuarto de corona circular).

  - En caso de que la x_{obj} sea mayor que la x_{IR}.

$$x_{supzs} \leq x_{obj} \leq x_{supzla} \quad \textbf{(3.100)}$$

  - En caso de que la x_{obj} sea menor que la x_{IR}.

$$x_{supzs} \geq x_{obj} \geq x_{supzla} \quad \textbf{(3.101)}$$

Para la representación de las curvas de nivel, se deben obtener los valores de los radios que están comprendidos entre los radios de la zona de seguridad y la zona de limitación de alturas. Estos radios son determinados a partir de incrementos de profundidad de 5m multiplicándolo por la pendiente de la zona de limitación de alturas y agregándolo al valor de radio anterior, el primer valor es el radio de la zona de seguridad.

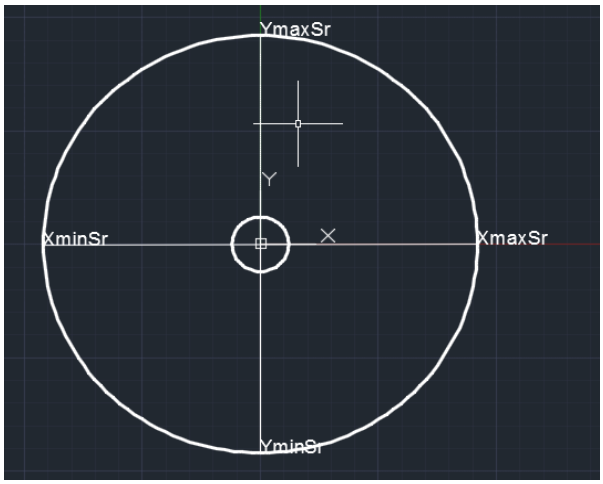

Figura 3.10 Límites superficiales de una servidumbre radioeléctrica.

3.3.5.1 Servidumbre radioeléctrica ILS

Las servidumbres de la ILS, tienen un diseño peculiar para el localizador y el equipo de planeo, a diferencia del resto de las instalaciones radioeléctricas poseen una servidumbre con forma de corona circular.

Todos los procedimientos para la determinación de intersecciones aplicadas en ambas superficies de la ILS siguen los mismos pasos que en los utilizados para la Superficie de Aproximación y de Transición.

El proceso de obtención de las curvas de nivel es bastante similar al procedimiento usado la Transición (para las superficies superior e inferior (ILS/LOC)) y a la Aproximación (para las superficies laterales (ILS/LOC y zona de limitación de alturas ILS/GP)).

Los métodos empleados para el análisis de intersecciones y la obtención de las matrices para la representación son *igpint*, *ilocint*, *radioint, GetIGp, GetILoc, GetSRadE*.

3.3.5.1.1 Servidumbre radioeléctrica ILS/LOC

A continuación, se especifican las expresiones utilizadas para determinar los límites superficiales.

$$X_{minILOC} = X_{IR} - 2500 \quad \textbf{(3.102)}$$

$$X_{maxILOC} = X_{IR} + 2500 \quad \textbf{(3.103)}$$

$$Y_1 = Y_{IR} - |d| * \tan 30 \quad \textbf{(3.104)}$$

$$Y_2 = Y_{IR} + |d| * \tan 30 \quad \textbf{(3.105)}$$

$$Y_{minILOC} = Y_{IR} - 2500 * \tan 30 \quad \textbf{(3.106)}$$

$$Y_{maxILOC} = Y_{IR} + 2500 * \tan 30 \quad \textbf{(3.107)}$$

$$Y_{ar} = Y_{IR} + 500 \quad \textbf{(3.108)}$$

$$Y_{ab} = Y_{IR} - 500 \quad \textbf{(3.109)}$$

$$X_{11} = X_{IR} + 500 * \tan 30 \quad \textbf{(3.110)}$$

$$X_{22} = X_{IR} - 500 * \tan 30 \quad \textbf{(3.111)}$$

- Aterrizaje por la izquierda.

$$X_1 = X_{pista} - \frac{l}{2} + posub \quad \textbf{(3.112)}$$

$$X_2 = X_{IR} - |X_1 - X_{IR}| \quad \textbf{(3.113)}$$

$$d = |X_1 - X_{IR}| \quad \textbf{(3.114)}$$

- Aterrizaje por la derecha.

$$X_2 = X_{pista} + \frac{l}{2} - posub \quad \textbf{(3.115)}$$

$$X_1 = X_{IR} - |X_{IR} - X_2| \quad \textbf{(3.116)}$$

$$d = |X_{IR} - X_2| \quad \textbf{(3.117)}$$

Donde:

- d, es la distancia que hay desde el umbral hasta la instalación radioeléctrica.
- Y_1, Y_2, X_1 y X_2 son los límites de la zona de seguridad.
- Y_{ar}, Y_{ab}, X_{11} y X_{22} son los límites de la zona de limitación de alturas que se extienden por encima de la instalación radioeléctrica.
- posub, es la distancia desplazado del umbral.
- l, es la longitud de la pista.

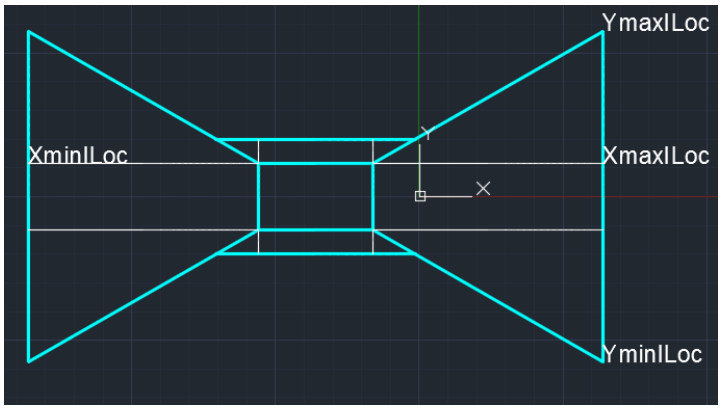

Figura 3.11 Límites superficiales de una servidumbre radioeléctrica ILS/LOC.

3.3.5.1.2 Servidumbre radioeléctrica ILS/GP

A continuación, se especifican las expresiones utilizadas para determinar los límites superficiales.

$$Y_1 = Y_{IR} + (|Y_{IR} - Y_{pista}| + \frac{a}{2} + 200) \quad \textbf{(3.118)}$$

$$Y_2 = Y_{IR} - (|Y_{IR} - Y_{pista}| + \frac{a}{2} + 200) \quad \textbf{(3.119)}$$

- En caso de que $X_{IR}>X_{pista}$.

$$X_2 = X_{IR} + \left|X_{pista} - \frac{l}{2} + posub\right| + 600 \quad \textbf{(3.120)}$$

$$X_1 = X_{IR} - \left|X_{pista} - \frac{l}{2} + posub\right| - 200 \quad \textbf{(3.121)}$$

$$X_{minIGP} = X_1 \quad \textbf{(3.122)}$$

$$X_{maxIGP} = X_{IR} + 5000 + |Y_{IR} - Y_{pista}| \quad \textbf{(3.123)}$$

$$Y_{maxIGP} = Y_{IR} + |X_{maxIGP} - X_{IR}| * \tan 20 \quad \textbf{(3.124)}$$

$$Y_{minIGP} = Y_{IR} - |X_{maxIGP} - X_{IR}| * \tan 20 \quad \textbf{(3.125)}$$

- En caso de que $X_{IR}<X_{pista}$.

$$X_1 = X_{IR} - |X_{pista} - \frac{l}{2} + posub| - 600 \quad \textbf{(3.126)}$$

$$X_2 = X_{IR} + \left|X_{pista} - \frac{l}{2} + posub\right| + 200 \quad \textbf{(3.127)}$$

$$X_{maxIGP} = X_2 \quad \textbf{(3.128)}$$

$$X_{minIGP} = X_{IR} - 5000 - |Y_{IR} - Y_{pista}| \quad \textbf{(3.129)}$$

$$Y_{maxIGP} = Y_{IR} + |X_{minIGP} - X_{IR}| * \tan 20 \quad \textbf{(3.130)}$$

$$Y_{minIGP} = Y_{IR} - |X_{minIGP} - X_{IR}| * \tan 20 \quad \textbf{(3.131)}$$

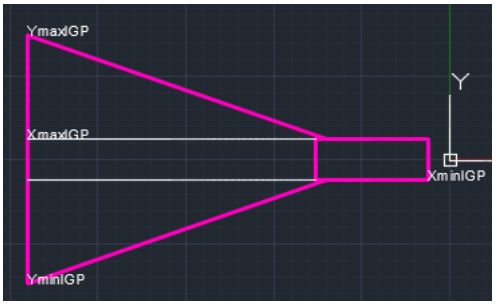

Figura 3.12 Límites superficiales de una servidumbre radioeléctrica ILS/GP.

3.4 Traslación de los puntos

La orientación de las pistas, vienen referenciadas respecto al norte magnético por lo cual los ángulos siguen la disposición que se muestra en la siguiente figura.

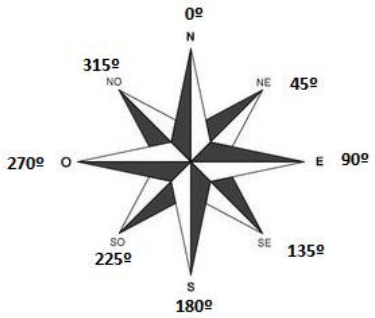

Figura 3.13 Orientación respecto al norte magnético.

Debido a que los métodos explicados en los anteriores apartados, solo son válidos para el caso en el cual las cabeceras son horizontales, es decir que la orientación de cada cabecera sea 09 y 27. Con la finalidad de simplificar los procesos de determinación de intersecciones (entre las servidumbres y los obstáculos), además de la representación de las servidumbres. Se trasladan las coordenadas del centro de la pista y de los objetos de forma que se tiene una pista horizontal.

La representación de las servidumbres, se realiza mediante un nuevo giro con el mismo ángulo, pero en sentido inverso de tal manera se obtiene la información necesaria para representar el contorno de las servidumbres con la orientación original.

Para realizar la traslación primero hay que determinar el ángulo del giro, este ángulo viene dado por la diferencia entre un ángulo α y 90º, donde α es el valor del ángulo de la cabecera que tenga una orientación menor respecto al norte magnético. Consecuentemente, α está situado siempre en el primero o en el cuarto cuadrante, por lo cual, su valor está dentro de un rango de valores comprendido entre 0º-180º (exclusive).

Por otro lado, es importante saber que dependiendo del sentido del giro el valor del ángulo dentro de las fórmulas del giro es negativo en caso de que se gire en sentido de las agujas del reloj y positivo en caso de que se realice en el giro en sentido opuesto a las agujas del reloj. De tal forma que, el valor del ángulo de giro (β) es igual a:

- En caso de que el ángulo se halle en el primer cuadrante (α < 90º).

$$\beta = -(90º - \alpha) \quad \textbf{(3.132)}$$

- En caso de que el ángulo se halle en el cuarto cuadrante (α > 90º).

$$\beta = \alpha - 90º \quad \textbf{(3.133)}$$

- En caso de que el valor de α sea igual a 90º el valor de β es igual a 0º debido a que estamos ante el caso en el que la pista es horizontal, por lo cual no es necesario que se realice ningún giro.

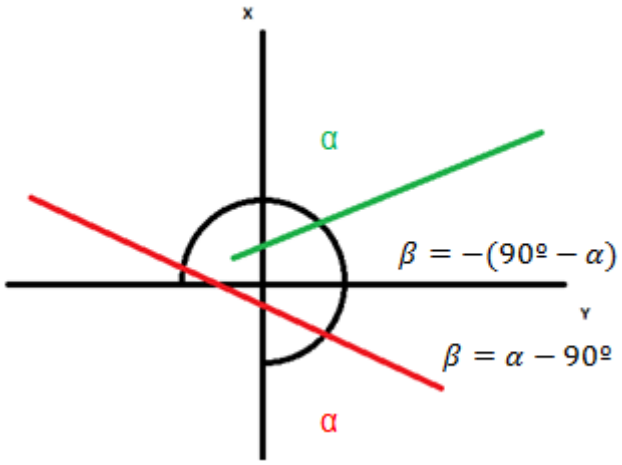

Figura 3.14 Determinación de ángulos para pistas no horizontales.

Sabiendo esto, aplicando las siguientes fórmulas obtenemos las nuevas coordenadas del punto a trasladar.

$$X' = X * \cos\beta - Y * \sin\beta \quad \textbf{(3.134)}$$

$$Y' = X * \cos\beta + Y * \sin\beta \quad \textbf{(3.135)}$$

Donde:

- X e Y son las coordenadas iniciales.
- X' e Y' son las nuevas coordenadas.
- B es el ángulo resultante entre la diferencia entre la orientación de la pista y 90º. [11]

En la *Figura 3.15* se puede apreciar, de manera visual lo que ocurre durante el proceso de giro.

Si suponemos de la orientación inicial es la de la imagen de la izquierda (pista no horizontal); sería necesario realizar el giro la pista y los obstáculos (puntos rosa) para poder hallar de manera más simple los puntos que conforman las servidumbres y definir si existen intersecciones.

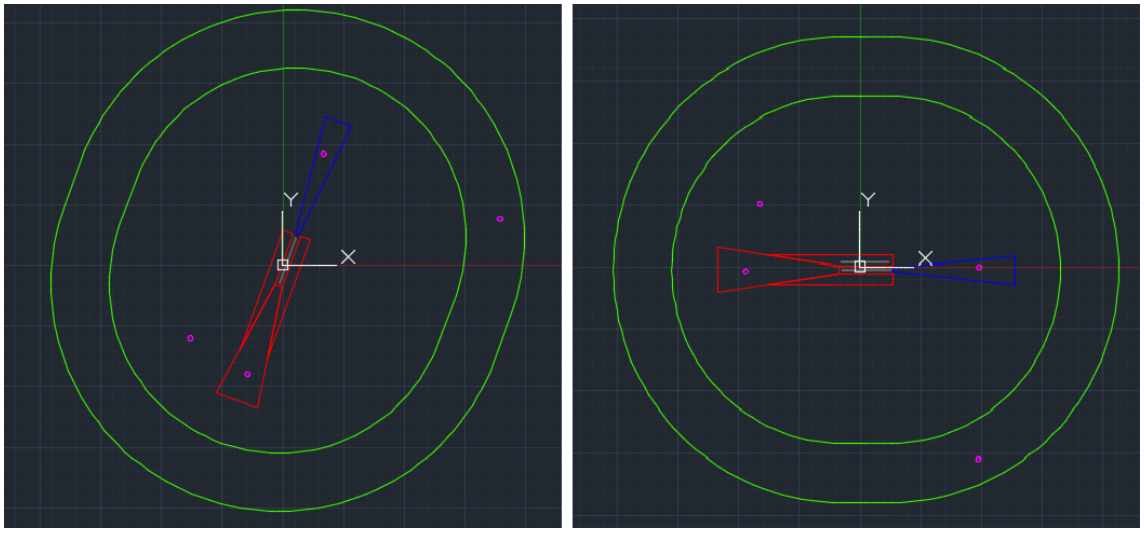

Figura 3.15 Representación de servidumbres con las mismas características dimensionales, pero con diferente orientación.

Una vez encontrados los puntos que conforman las servidumbres, es necesario que se realice la traslación inversa para obtener las coordenadas con la orientación original de la pista, las cuales se determinan a partir de las siguientes expresiones:

$$X = Y' * \sin\beta + X' * \cos\beta \quad \textbf{(3.136)}$$

$$Y = Y' * \cos\beta - X' * \sin\beta \quad \textbf{(3.137)}$$

Donde:

- X e Y son las coordenadas con la orientación original.
- X' e Y' son las coordenadas del punto con orientación horizontal.
- B es el ángulo resultante entre la diferencia entre la orientación de la pista y 90º.

Ambos procesos de traslación son llevados a cabo por los métodos *giro* y *giroinv* programados dentro de la librería de clases *Intersecciones*.

3.5 Procesamiento de los parámetros de entrada

Uno de los primeros pasos que el usuario debe realizar, para poner en funcionamiento la simulación de las servidumbres, es la de la introducción de las características de la pista. Así como también, la introducción de la información sobre los posibles obstáculos y las instalaciones radioeléctricas.

Por lo tanto, la aplicación debe ser capaz de identificar si se han introducido todos los datos y con el formato correcto. En caso de que se detecte alguna anomalía en los datos introducidos (formato incorrecto, por ejemplo si al introducir el valor de la longitud de la pista introducimos una letra) o que no se hayan introducido todos los datos requeridos, el programa muestra un mensaje de advertencia para que el usuario corrija el error o que rellene los datos que le faltan por introducir.

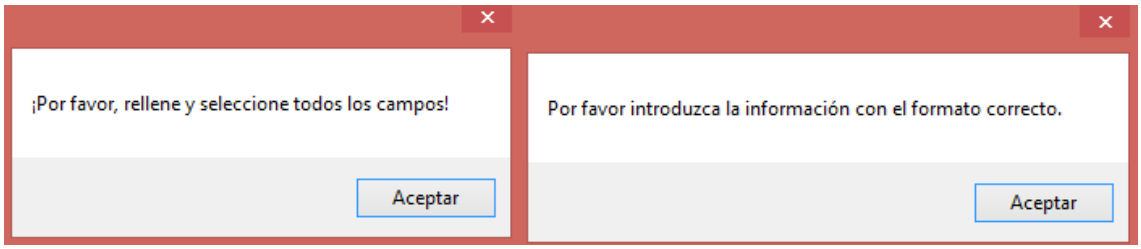

Figura 3.16 Mensajes de advertencia que el usuario verá en caso de que no se haya rellenado correctamente los parámetros.

Además, en caso de que el usuario quiera ejecutar alguna función sin haber introducido previamente los datos requeridos el programa también inicializa un aviso para que el usuario inserte la información necesaria.

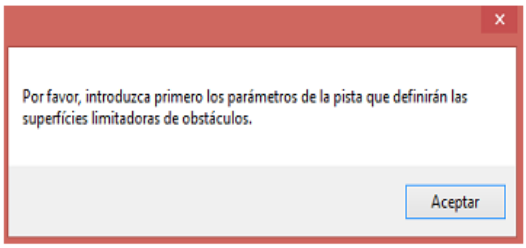

Figura 3.17 Mensaje de advertencia que el usuario verá en caso de que se intente ejecutar alguna función sin haber introducido previamente los parámetros.

Una vez que el usuario haya introducido correctamente, el programa procesa todos los parámetros para saber cuáles son las características dimensionales de las servidumbres, por ejemplo si se ha introducido como dato que la pista tiene una longitud de pista de 1000m la aplicación identifica que la pista es de clave 2.

3.6 Compatibilidad de datos

A continuación, el programa determina si los parámetros introducidos son compatibles entre ellos, esto significa que existen ciertas combinaciones que de acuerdo a la normativa (al Anexo 14 de la OACI, en este caso) que no existen. Consecuentemente, el programa muestra un aviso para que el usuario pueda rectificar los parámetros de entrada. Por ejemplo, si nos fijamos en la siguiente figura, se puede observar que para pistas de clave 1 y 2 no pueden tener letras de clave D, E y F; por lo tanto, si al introducir los parámetros introducimos por ejemplo una pista de longitud 1000m (clave 2) con letra de clave D, E o F nos saldría la siguiente advertencia.

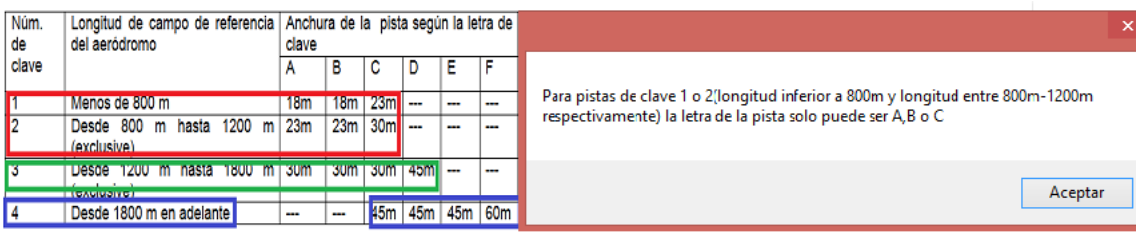

Figura 3.18 Mensaje de advertencia que el usuario verá en caso incompatibilidad (izquierda); Incompatibilidades de diseño de pista (derecha).

En la *Figura 3.18*, se pueden apreciar todas las incompatibilidades que existen entre longitudes de pista y claves de pista. Pero, existen también incompatibilidades entre la clave de la pista (relacionado directamente con la longitud de la pista) con el tipo de aproximación para el cual está diseñado la pista. Como bien se puede apreciar en la *Figura 3.19*, la incompatibilidad tiene lugar cuando se intenta introducir una pista que no sea clave 3 o 4 (menor de 1200m) para una pista diseñada para aproximaciones de precisión de categoría II o III (cuadro verde).

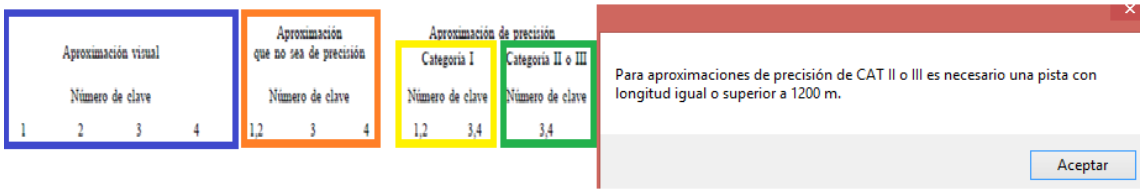

Figura 3.19 Mensaje de advertencia que el usuario verá en caso incompatibilidad (izquierda); Incompatibilidades de aproximación (derecha).

La compatibilidad es un punto bastante importante, porque dependiendo de los parámetros introducidos las servidumbres tienen unas características dimensionales específicas, además dependiendo de la combinación algunas pistas no poseen todas las SLO. Es el caso de la superficie de aproximación interna, transición interna y la superficie de aterrizaje interrumpido donde solo se implementan, cuando estemos con una pista que esté preparado para

aproximaciones de precisión tanto de categoría I, II o III. De la misma manera, todas las pistas que no sean de precisión o de no precisión y de clave 3 o 4, su superficie de aproximación se dotará únicamente de una primera sección.

3.7 Introducción y modificación de la lista de obstáculos y la lista de instalaciones radioeléctricas.

De la misma manera que en la introducción de los parámetros de la pista, el programa analiza y procesa los datos referidos a los obstáculos y a las servidumbres radioeléctricas.

De tal forma, con el objetivo de dar más robustez a la aplicación, se examina la información introducida asegurando que el formato introducido sea correcto.

A diferencia de los parámetros de la pista, los obstáculos y las instalaciones poseen una identificación para que el usuario pueda controlar mejor los datos con los que está trabajando. Por lo cual, el programa además de comprobar que la introducción de los datos ha sido correcta, también se asegura de que no exista una identificación duplicada, si se da la situación de que exista una identificación duplicada en el caso de los obstáculos directamente se eliminaría la última línea que genere el conflicto; en el caso de las instalaciones radioeléctricas el aplicativo mostraría un aviso para alertar al usuario.

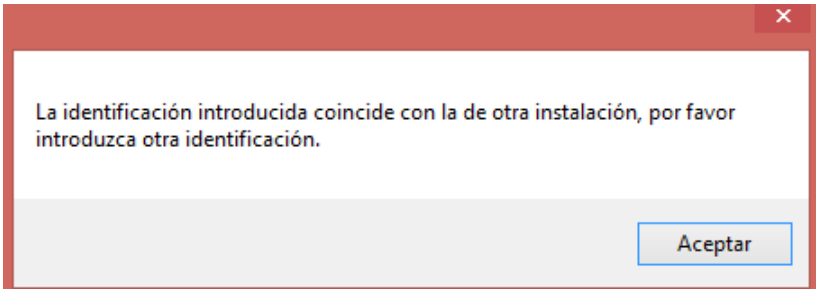

Figura 3.20 Mensaje de advertencia que indica duplicación de identificadores.

Otra de las funcionalidades que ofrece el programa es el de modificar la información sobre obstáculos e instalaciones que se haya introducido previamente.

Para ello, el aplicativo comprueba primero si ya se ha introducido anteriormente la información. En caso de que sea afirmativo, se permite la posibilidad de eliminar (total o parcialmente) la lista, modificar información previa o introducir nueva información.

Por otro lado, si se da la situación de que se está utilizando la opción de *Eliminar* datos, el programa controla los valores eliminados sabiendo cuando se han eliminado, para advertir al usuario de que ya no hay más información que borrar y que para poder ejecutar las funciones de intersección (tanto físicas como radioeléctricas) o representación de servidumbres físicas, es necesario la introducción de nuevos valores.

4.Conclusiones

Las servidumbres aeronáuticas se definen con el fin de mantener la seguridad operacional dentro de la infraestructura aeroportuaria. Por ello, se implementan un conjunto de SLO y servidumbres radioeléctricas, que limitan la profundidad de los objetos presentes en el entorno aeroportuario.

Una vez diseñadas las servidumbres, se deberá revisar que no emerjan nuevos obstáculos que interfieran en la seguridad operacional. En caso de que sea necesario el emplazamiento de un objeto, que puede vulnerar las servidumbres, se deberá informar al Ministerio del Aire para que analice el caso y autorice la inclusión.

Por lo cual, el objetivo de este trabajo ha sido desarrollar una aplicación, que tenga la finalidad de simplificar los procesos de diseño de las servidumbres y el posterior análisis de obstáculos.

La normativa con la cual la aplicación diseña las servidumbres son el *Anexo 14 de la OACI* y el *Decreto 584/1972 de 24 de Febrero de la BOE.*

El simulador se ha desarrollado con lenguaje C# en *Microsoft Visual Studio 2012*. A partir del cual, se han diseñado un conjunto de *Forms* con los cuales el usuario podrá interactuar (introducción, modificación de parámetros) y ejecutar todas las funcionalidades del programa.

El programa funciona de manera *Out-of-Process,* debido a que se compila independientemente y, además ejecuta el aplicativo de *AUTOCAD* de forma externa.

La visualización del análisis de obstáculos es llevado a cabo por los *Forms*, por otro lado, la representación de las servidumbres es realizada conjuntamente entre los *Forms* y *AUTOCAD*. Además de estas funcionalidades, la aplicación también es capaz de analizar los parámetros introducidos, informando al usuario cuando éstos induzcan a error (errores de formato o compatibilidad).

El procedimiento diseñado en la determinación de obstáculos, se basa en la sectorización de las diferentes servidumbres, una vez se haya comprobado que el objeto esté dentro de los límites de la servidumbre, se identifica el sector en el cual está el objeto. Finalmente, se analiza si la profundidad del objeto vulnera la profundidad de la superficie, en caso afirmativo existe intersección y el objeto es considerado como un obstáculo.

A partir de las coordenadas de los límites superficiales, además de los datos de la pista, la franja y las características superficiales. Se determina la información necesaria para representar las curvas de nivel que serán enviadas a *AUTOCAD*.

Para simplificar los procedimientos de análisis, se realiza una translación de los puntos de referencia de la pista y los objetos, así quedan con una orientación horizontal respecto al norte magnético.

En caso de que se cargue un plano, es importante que el sistema de coordenada de los parámetros sea el mismo que el del plano, ya que se introducen en el plano de forma georreferenciado; evitando así que las servidumbres queden situadas erróneamente en el plano.

La aplicación cumple con todas las funciones fijadas en los objetivos, aunque, a pesar de ello la aplicación queda disponible para implementar y añadir futuras mejoras y ampliaciones que mejoren la funcionalidad del programa, como ahora:

- Optimización del código, de manera que la ejecución sea más eficiente y consuma menos recursos de computación.
- Implementar la lectura de un archivo de objetos de un fichero .xls, de manera que no sea necesario introducir a mano los datos.
- Permitir que se cargue un fichero de plano desde la interfaz de usuario, sin tener que realizarlo desde *AUTOCAD*.
- Desarrollar los métodos necesarios para que se representen las servidumbres en diferentes capas, así como también corregir la orientación de los letreros de las curvas de nivel.
- Adaptar los métodos, de manera en la cual se puedan trabajar con los datos de más de 1 pista.

5.Bibliografía

[1] “Anexo 14, OACI”.

[2] “Real Decreto 584/1972 de 24 de Febrero, BOE”.

[3] Marcos García Cruzado., “Entorno orográfico”, Cap.7 en Ingeniería aeroportuaria, pp. 223-242, ETSI DE INGENIEROS AERONÁUTICOS.

[4] http://www.elmundo.es/elmundo/2007/07/18/internacional/1184714515.html

[5] http://www.aeronauticos.org

[6] http://www.elmundo.es/elmundo/2005/10/24/sociedad/1130169434.html

[7] http://blog.cadnex.com/post/AutocadProgramacionOpciones.aspx

[8]http://docs.autodesk.com/ACD/2010/ENU/AutoCAD%20.NET%20Developer%27s%20Guide/

[9] http://blog.gbellmann.technology/2010/10/24/primer-programa-en-autocad-con-net/

[10] Fco. Javier Ceballos., “Microsoft C#. CURSO DE PROGRAMACIÓN”, Ra-Ma.

[11] Carlos Ivorra Castillo., “Geometría”.

[12] “Datos del Aeródromo de Almería-LEAM, AIP España”.

[13] http://www.synnatschke.de/geo-tools/coordinate-converter.php

[14] “Real Decreto 1837/2009 de 27 de Noviembre, BOE”.

[15] “Planos de servidumbres de aeródromo y radioeléctricas, Distribución de Hojas, Real Decreto 1837/2009 de 27 de Noviembre, BOE”.

[16] “Planos de servidumbres de aeródromo y radioeléctricas, Hoja 11, Plano 2, Real Decreto 1837/2009 de 27 de Noviembre, BOE”.

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

Título: Automatización del diseño de las superficies limitadoras de obstáculos

Titulación: Grado en Ingeniería de Aeropuertos

Autor: Wenhao Cheng

Director: Óscar Maldonado Díaz

Data: 30 de Junio de 2015

ÍNDICE

ANEXO A. Glosario 5
ANEXO B. Manual de usuario 3
ANEXO C. Ejecución del simulador con un caso real 12
ANEXO D. Código de la aplicación 21

Índice de figuras

Figura B.1 Carpeta donde está localizado el archivo *.exe*……..3
Figura B.2 Ventana principal del simulador……...3
Figura B.3 Opciones disponibles de a escoger, tipo de aproximación (izquierda); letra de clave (derecha)……4
Figura B.4 Opciones disponibles en la lista desplegable *Funciones*…….5
Figura B.5 Ventanas que se inicializarán para la introducción de la información requerida……..6
Figura B.6 Ventana *Obstáculos*……..……7
Figura B.7 Ventana *Datos sobre la pista e intersecciones con las SLO*……...8
Figura B.8 Ventana *Servidumbres radioeléctricas*……9
Figura B.9 Tipos de antenas disponibles……....9
Figura B.10 Ventana *Instalaciones radioeléctricas*……10
Figura B.11 Ventana *Servidumbres radioeléctricas-Intersecciones*……11
Figura C.1 Características físicas de la pista……..12
Figura C.2 Instalaciones radioeléctricas que se introducirán como parámetros……13
Figura C.3 Convertidor de coordenadas……13
Figura C.4 Situación global de los objetos……...14
Figura C.5 Emplazamiento de los objetos 1(izquierda) y 2 (derecha)……15
Figura C.6 Emplazamiento de los objetos 3 (izquierda) y 4 (derecha)……15
Figura C.7 Emplazamiento del objeto 5……16
Figura C.8 Emplazamiento de los objetos 6 (izquierda) y 7 (derecha)……...16
Figura C.9 Emplazamiento de los objetos 8 (izquierda) y 9 (derecha)……16
Figura C.10 Diseño real (abajo) y diseño simulado (arriba)……17
Figura C.11 Datos para la configuración: aterrizajes por cabecera 07 y despegues por cabecera 25……...18
Figura C.12 Diseño simulado con las 2 configuraciones de pista……19
Figura C.13 Detalle de superposición de superficies en el diseño simulado y en el diseño real……19

Índice de tablas

Tabla C.1. Conversión a coordenadas UTM……...13
Tabla C.2. Lista de objetos a analizar……..14

ANEXO A. Glosario

A

AESA (Agencia Estatal de Seguridad Aérea), es una organización encargada de la seguridad de la aviación civil en el ámbito territorial de España.

AIP (Publicación de información aeronáutica), conjunto de documentos editados por las autoridades competentes en aviación civil, que contiene información aeronáutica de carácter esencial para la navegación aérea.

B

BOE (Boletín Oficial del Estado), es el diario oficial del Estado español dedicado a la publicación de determinadas leyes, disposiciones y actos de inserción obligatoria.

C

CWY (Zona Libre de Obstáculos), porción de superficie situado al final de la pista en la cual no se puede instalar ningún elemento.

D

DME (del inglés: Distance Measuring Equipment), es un sistema electrónico que permite establecer la distancia entre éste y una estación emisora.

G

GP (del inglés: Glide Path), es una antena trasmisora dedicada a la senda del planeo, se sitúa a un lado de la zona de la pista donde se produce la toma de señales.

I

ILS (Sistema de Aterrizaje Intrumental), es el sistema de ayuda a la aproximación y el aterrizaje, el cual permite el control del guiaje de una aeronave durante las fases de aproximación.

IMC (del inglés: Instrument Meteorological Conditions), es una categoría de vuelo, que describe las condiciones climatoológicas requeridas para que un piloto vuele por instrumentos.

L

LOC (Localizador), son una serie de antenas direccionales situadas al final de pista, se encarga de proporcional la guía lateral a las aeronaves durante las fases de aproximación.

O

OACI (Organización de Aviación Civil Internacional), es una agenia que tiene por objeto, el estudio de los problemas de la aviación civil internacional, además de promover los reglamentos y normas únicos en la aeronáutica mundial.

OCA/H, es la elevación mínima por encima de la elevación del umbral, a la cual debe comenzar el procedimiento de aproximación frustrada.

S

SLO (Superficie Limitadora de Obstáculos), son los límites definidos en el entorno aeroportuario, que tendrán el objetivo de salvaguardar la eficiencia y la seguridad operacional de las aeronaves.

U

UTM (del inglés: Universal Transverse Mercator), es un sistema de coordenadas basado en la proyección cartográfica transversa de Mercator.

V

VMC (del inglés: Visual Meteorological Conditions), son las condiciones de climáticas con las cuales se puede realizar un vuelo visual.

VOR (del inglés: VHF Omnidirectional Radio Range), radioayuda que utilizan las aeronaves en la navegación, con la cual pueden seguir una ruta preestablecida.

ANEXO B. Manual de usuario

En este anexo, se explicará la manera en la cual el usuario debe interactuar con el programa, mostrando las diferentes pantallas. Se especificarán también, los parámetros que se tienen que introducir así como también las funciones que ofrece cada pantalla.

En primer lugar, para inicializar el simulador será necesario darle doble clic al icono de *Simulador* dentro de la carpeta donde tenemos guardado todo los ficheros relacionados con el código y la aplicación.

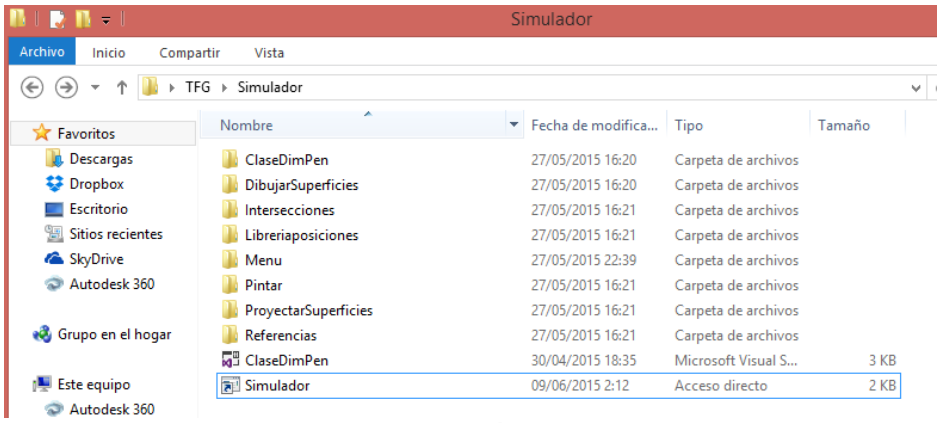

Figura B.1 Carpeta donde está localizado el archivo *.exe*.

Ventana Simulador SLO

Una vez hayamos inicializado el programa, se muestra la ventana principal del simulador, la cual podemos apreciar en la siguiente figura, en ella se han destacado 3 partes, cada cual tiene funciones diferentes.

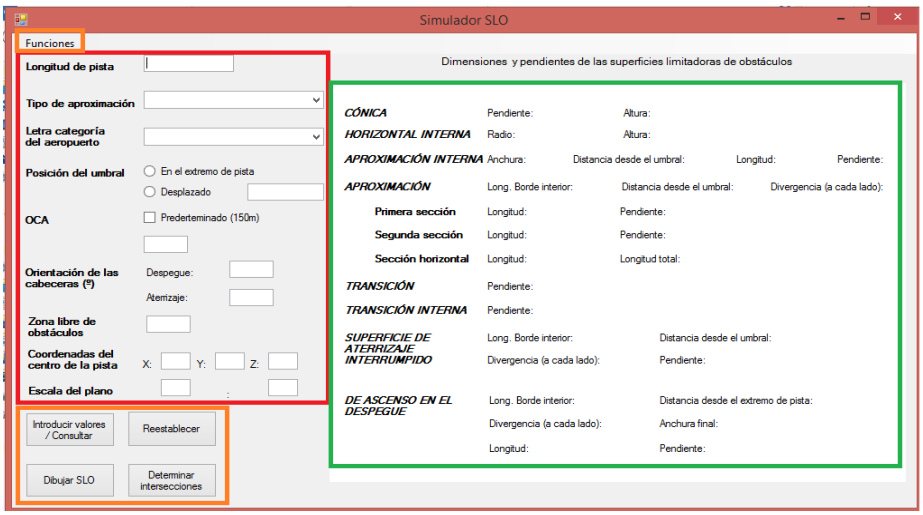

Figura B.2 Ventana principal del simulador.

La parte dentro del cuadrado rojo, son los parámetros de la pista para la cual queremos diseñar las servidumbres aeronáuticas. Los parámetros son los siguientes:

- Longitud de la pista en metros.
- Tipo de aproximación para la cual está diseñada la pista, este parámetro se introduce seleccionando una opción de una lista desplegable que tiene las opciones pre configuradas.
- Letra de categoría de la pista, al igual que en el anterior caso el parámetro se introduce seleccionando una opción de una lista desplegable.

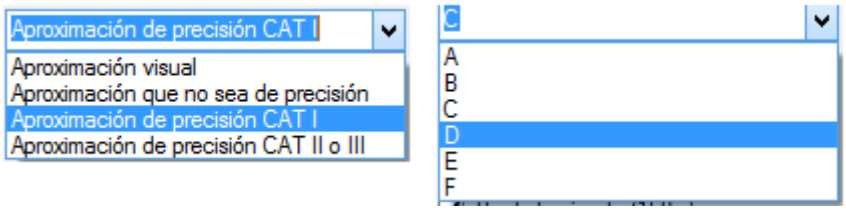

Figura B.3 Opciones disponibles de a escoger, tipo de aproximación (izquierda); letra de clave (derecha).

- Posición del umbral, en este parámetro tenemos que indicar los metros a los que está desplazado el umbral en caso de que el umbral no esté desplazado (esté situado en el extremo de la pista) seleccionamos la opción *En el extremo de pista*.
- OCA es la altitud de franqueamiento de obstáculos en m. En caso de que se vaya a introducir un valor diferente al predeterminado (150m) es necesario que este valor sea superior a los 150m.
- Orientación de las cabeceras en grados, en este parámetro al introducir la orientación de una cabecera automáticamente el programa determina la orientación la otra cabecera, el rango de valores esta entre 0º-359º inclusive, al introducir 360º el programa automáticamente transforma el valor introducido en 0º.
- Zona libre de obstáculos, en este parámetro hay que indicar los metros a partir del extremo de pista, en el cual esté situada la zona libre de obstáculos, en caso de que no exista zona libre de obstáculos o está esté situada en el extremo de pista hay que introducir 0.
- Coordenadas del centro de la pista.
- Escala al cual queremos que se representen las servidumbres en *AUTOCAD*.

En caso de que algún parámetro se haya introducido incorrectamente el programa indicará el error para que el usuario lo introduzca de nuevo correctamente *(ver Solución adoptada / Procesamiento de los parámetros de entrada)*.

En los cuadros naranjas, están las funciones que se pueden ejecutar en esta ventana. Antes de nada, para poder cargar cualquier otra función es necesario haber rellenado los parámetros, además de ejecutar posteriormente el botón *Introducir valores/Consultar* a partir del cual se obtendrán las características dimensionales de las servidumbres físicas (que se mostrarán en el cuadro verde). Después de estos 2 pasos, podremos ejecutar cualquier otro botón u opción de la lista desplegable *Funciones.* A continuación, se explicarán la función del resto de los botones y las opciones de la lista desplegable *Funciones*:

- El botón *Reestablecer,* sirve para borrar todos los datos introducidos y volver al punto de partida.
- El botón *Dibujar SLO*, sirve para iniciar *AUTOCAD* y representar todas las servidumbres físicas.
- El botón *Determinar intersecciones*, sirve para determinar la existencia de alguna intersección entre las SLO y una lista de objetos.

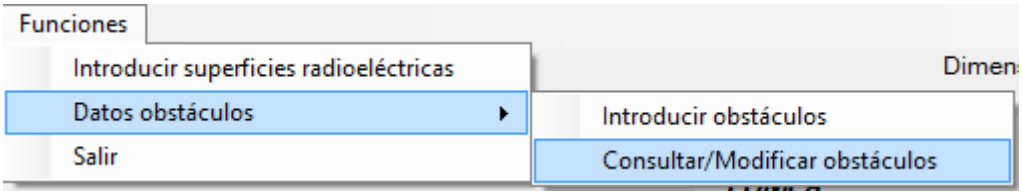

Figura B.4 Opciones disponibles en la lista desplegable *Funciones*.

- La opción *Introducir superficies radioeléctricas*, inicializa la ventana *Servidumbres radioeléctricas* en la cual podríamos introducir los datos de las instalaciones radioeléctricas que haya en el recinto aeroportuario así como en sus aproximaciones y así determinar las servidumbres radioeléctricas.
- La opción *Introducir obstáculos,* inicializa la ventana *Obstáculos* en la cual se introducen los datos de los objetos que queremos determinar si son obstáculos o no.
- La opción *Consultar/Modificar obstáculos*, estará disponible una vez que se hayan introducido los datos de los objetos. Al clicar esta opción se inicializa la ventana *Obstáculos* con los datos de los objetos que se hayan introducido anteriormente. En caso de que no se haya cargado previamente ningún objeto la aplicación avisará al usuario cuando quiera ejecutar esta opción, para que introduzca primero los datos.
- La opción *Salir*, cierra el programa.

Como ya se ha mencionado, para poder usar cualquiera de las otras funciones exceptuando la de *Introducir obstáculos* es necesario que se hayan rellenado correctamente los parámetros y haber ejecutado el botón *Introducir valores/Consultar.*

Dentro del cuadro verde podemos apreciar las características dimensionales de las servidumbres físicas, al inicializar el programa los no se muestra ningún dato una vez.

Pistas que requieren más información

Para pistas de clave 3 o 4 (longitud igual o superior a 1200m) se requiere que indiquen si las características operacionales permiten una reducción en la pendiente de la Superficie de Ascenso en el despegue (el valor predeterminado es del 2%). Por otro lado, si la pista además de tener una longitud igual o superior está diseñada para operaciones que no sean de precisión, se pide al usuario que indique si tiene derrotas superiores a 15º en determinadas situaciones de vuelo.

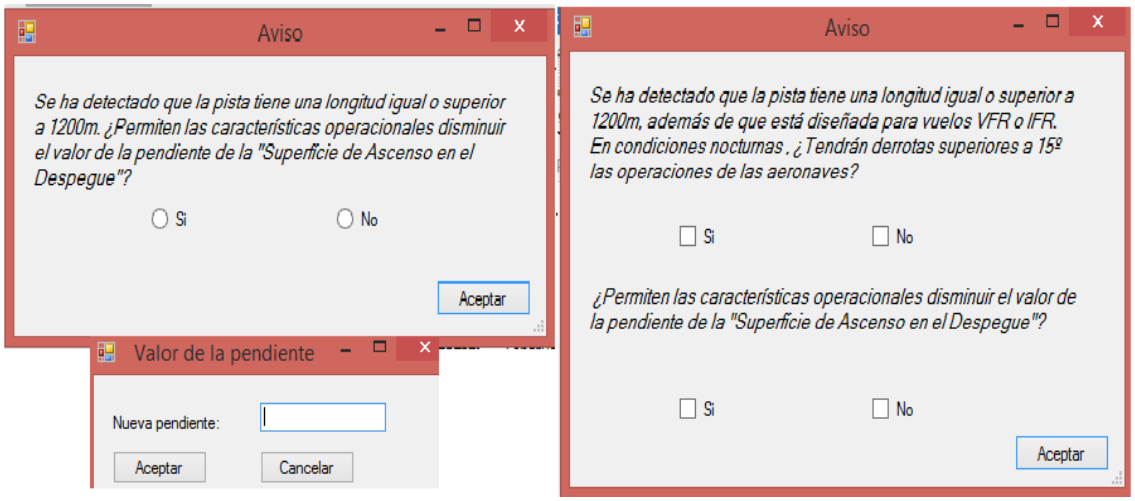

Figura B.5 Ventanas que se inicializarán para la introducción de la información requerida.

Es importante saber que la información que se introduzca varia las características dimensionales de la Superficie de Ascenso en el despegue (ver consideraciones de la *Figura 1.1.2*).

Ventana Obstáculos

En esta ventana se puede introducir, modificar o eliminar los datos sobre los diferentes objetos, de los cuales se quieren determinar si representan un obstáculo para el desarrollo seguro de las operaciones aéreas.

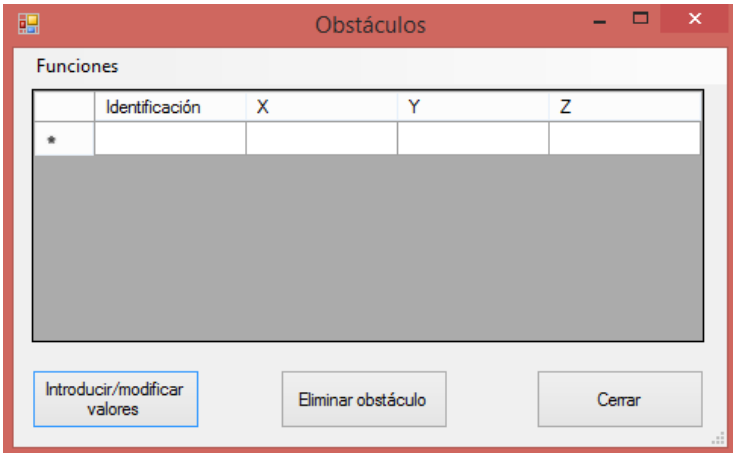

Figura B.6 Ventana *Obstáculos*.

Los datos de los objetos se introducen directamente en la malla de datos de la ventana, una vez que todos los datos hayan sido rellenados, es necesario pulsar el botón *Introducir/modificar valores* para que el programa procese los datos rellenados. Todas las filas de la malla que tengan alguna columna vacía serán ignoradas y no se introducirían en la lista de objetos, de la misma manera que si se repite una identificación, se conserva el objeto que se haya introducido en las filas anteriores.

Una vez ya se han introducido los valores, a partir del botón *Eliminar obstáculo* o la opción *Reestablecer* de la lista desplegable *Funciones* se pueden eliminar las filas de las celdas seleccionadas o toda la lista respectivamente.

En caso de que la ventana haya sido inicializada, a partir de una función para consultar o modificar los valores al inicializarse se podrá apreciar una malla rellenada con los datos de los objetos introducidos previamente. Como observación, es importante saber que en caso de que ya exista una lista previa de datos y se inicializa la ventana con la opción *Introducir obstáculos,* se reiniciaría la lista de objetos y sería necesario volver a introducir los valores.

Ventana Datos sobre la pista e intersecciones con las SLO

Como bien indica el nombre. En esta ventana, inicializada desde el botón *Determina Intersecciones* desde la ventana *Simulador SLO,* se muestran los parámetros de la pista como la longitud, la clave, la categoría, el tipo de aproximación y las coordenadas. De la misma manera que nos presenta también, los datos del obstáculo y nos indica si intersectan o no con las SLO en caso afirmativo nos indicarán también la profundidad *z* a la que se produce la intersección.

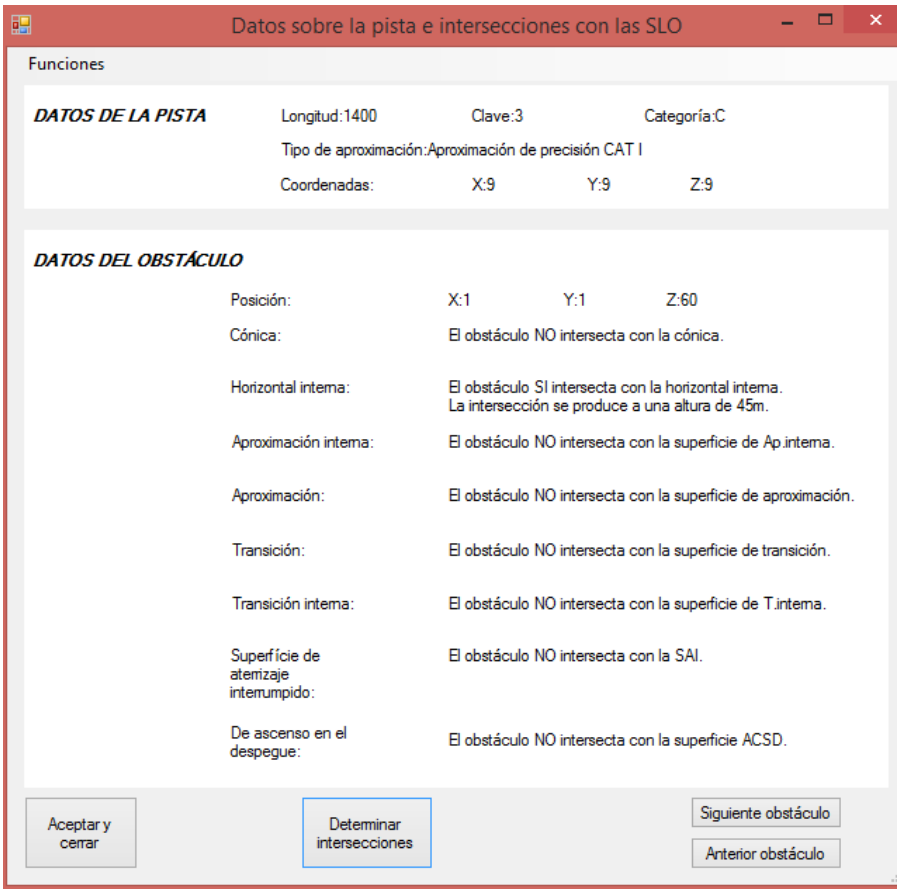

Figura B.7 Ventana *Datos sobre la pista e intersecciones con las SLO*.

En caso de que no se hayan introducido la lista de objetos, en la lista de opciones Funciones podemos encontrar las opciones de *Introducir obstáculos* y *Consultar/Modificar obstáculos*. Es fundamental, que haya una lista de objetos para poder ejecutar el botón *Determinar intersecciones* el cual será necesario clicar siempre que se produzca una modificación en la lista o cuando se introduzca una lista nueva.

Además, en caso de que la lista tenga más de un objeto aparecerán los botones *Siguiente obstáculo* y *Anterior obstáculo* a partir de los cuales podremos consultar la información del resto de los objetos.

Ventana Servidumbres Radioeléctricas

Esta ventana se inicializa a partir de la opción *Introducir superficies radioeléctricas* de la ventana *Simulador SLO* o del botón *Consultar/Modificar instalaciones* de la ventana *Servidumbres radioeléctricas-Instalaciones* que se explica en uno de los siguientes apartados, en esta ventana se encuentran todas las funciones relativas a las servidumbres radioeléctricas como ahora determinar las intersecciones o la representación en *AUTOCAD*.

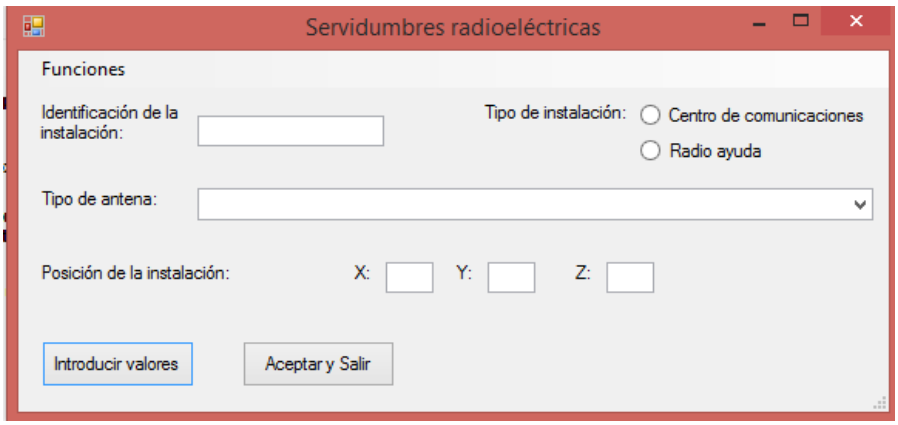

Figura B.8 Ventana *Servidumbres radioeléctricas*.

En primer lugar, es necesario rellenar la información de al menos una instalación radioeléctrica, por ello el programa pide:

- La identificación de la instalación, en caso de que se introduzca una identificación existente, la aplicación advertirá de ello para que se introduzca otro identificador.
- Tipo de instalación, dependiendo de la opción que se escoja se muestran un determinado conjunto de opciones para seleccionar el tipo de antena.

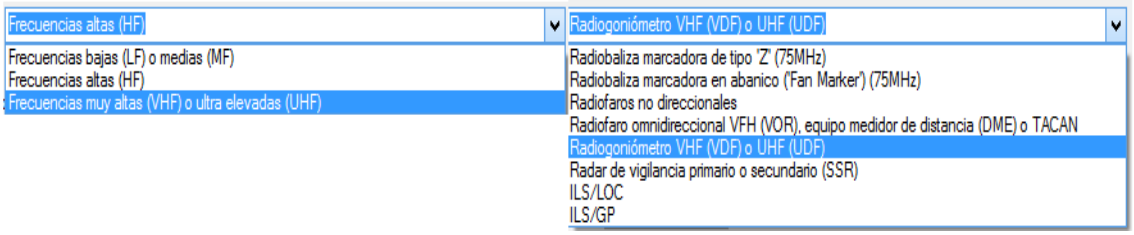

Figura B.9 Tipos de antenas disponibles.

- Tipo de antena, a escoger de la lista desplegable.
- Coordenadas de la instalación.

Las otras *Funciones* que ofrece esta ventana son:

- *Consultar/Eliminar instalación* en la cual se inicializa la ventana *Instalaciones radioeléctricas*.
- *Representar servidumbres* a partir de la cual inicializamos *AUTOCAD* y representamos todas las servidumbres de las instalaciones radioeléctricas introducidas.
- *Determinar intersecciones* a partir del cual se determinan las intersecciones de los objetos con las servidumbres radioeléctricas.

Cuando de clica esta opción se inicializa primero la ventana *Obstáculos* mostrando la información de una lista de objetos si se ha introducido previamente, en caso contrario es necesario que se introduzca al menos los datos de un objeto correctamente. Una vez se hayan introducido los objetos se inicializa la ventana *Servidumbres radioeléctricas – Intersecciones*.

Ventana Instalaciones radioeléctricas

Esta ventana se inicializa a partir de la opción *Consultar/Eliminar instalación* de la ventana *Servidumbres radioeléctricas*, en esta ventana se pueden apreciar la lista con las instalaciones que se han introducido en la anterior ventana. Además, está disponible la opción de eliminar las instalaciones (seleccionadas) o reestablecer los datos de la lista.

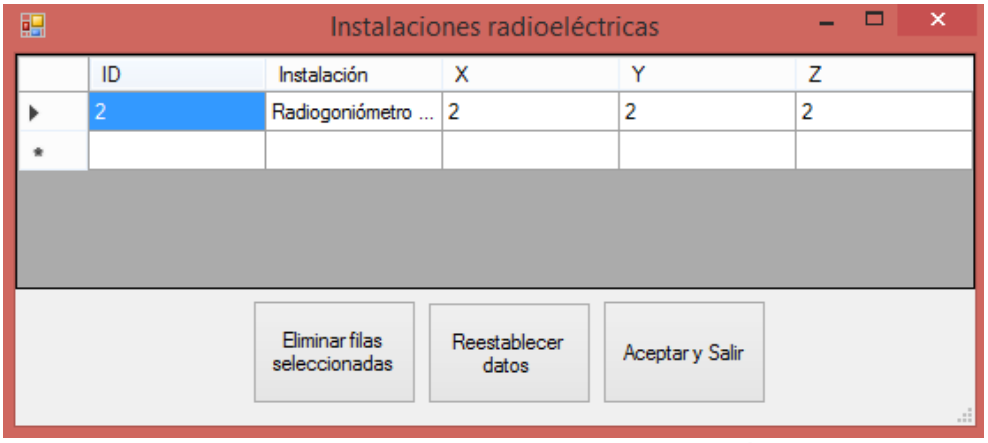

Figura B.10 Ventana *Instalaciones radioeléctricas*.

Ventana Servidumbres radioeléctricas-Instalaciones

Esta ventana se inicializa a partir de la opción *Determinar intersecciones* de la ventana *Servidumbres Radioeléctricas*, en este *Form* se muestran los datos de las instalaciones, además de todos los objetos indicando si estos intersectan o no y la profundidad a la que se producen las intersecciones, si ésta existe.
Observando la siguiente figura, se identifican los siguientes botones:

- *Consultar/Modificar instalaciones*, con el cual podremos modificar o ver todas las instalaciones introducidas anteriormente.
- *Consultar/Modificar obstáculos*, con el cual podremos modificar o ver los objetos introducidos previamente.
- *Siguiente instalación* y *Anterior instalación*, en caso de que la lista de instalaciones esté formado por más de una instalación a partir de estos

botones podremos ir consultando la información de las otras instalaciones.

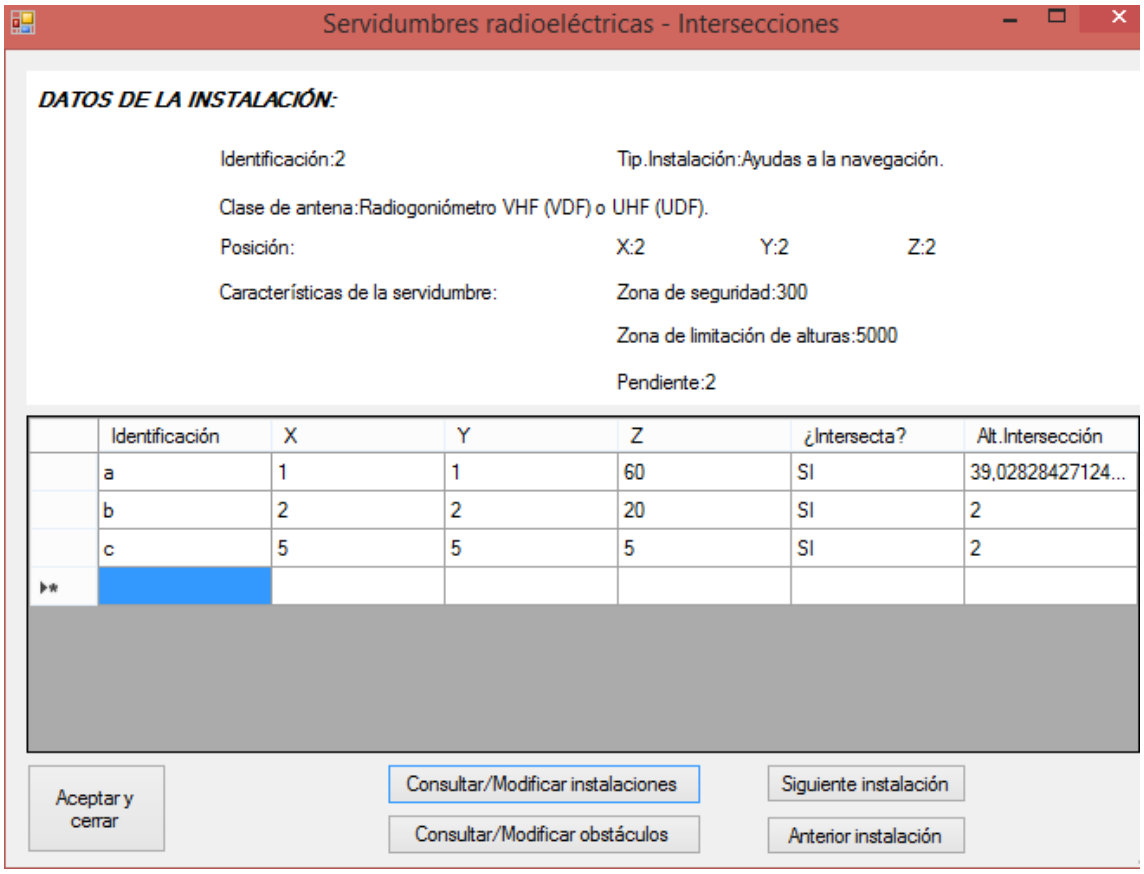

Figura B.11 Ventana *Servidumbres radioeléctricas-Intersecciones*.

ANEXO C. Ejecución del simulador con un caso real

En este anexo, se ejecutará la aplicación introduciendo los datos del aeropuerto de Almería como parámetros de entrada. Así como también, la información relativa de varías de las instalaciones radioeléctricas.

Una vez se hayan obtenido las características dimensionales de las servidumbres. Se introducirá una lista de objetos, con diferentes configuraciones, para comprobar si vulneran o no las servidumbres.

Finalmente, se representarán las servidumbres y se comparará el resultado obtenido con los planos reales de las servidumbres del aeropuerto de Almería.

Para simplificar la computación y facilitar el análisis de las representaciones, no se representarán las servidumbres de todas las configuraciones de pista posibles. De la misma manera, tampoco se introducirán todas las instalaciones radioeléctricas.

Parámetros de entrada [12][13][15]

Los parámetros de entrada, se obtienen a partir del AIP del aeropuerto de Almería. Se escogerá la configuración, con la cabecera 25 como pista de aterrizaje y la cabecera 07 como pista de despegue.

RWY	Orientación Direction	DIM (m)	THR PSN	THR ELEV TDZ ELEV	SWY (m)	CWY (m)	Franja (m) Strip (m)	OFZ	RESA (m)	RWY/SWY SFC PCN
07 (1)	073.01° GEO 074° MAG	3200 x 45	365024.4265N 0022308.3345W	THR: 8 m/26 ft TDZ: No	No	60 x 300	3320 x 300	No	190 x 150	RWY: Asfalto/Asphalt PCN 59/F/B/W/T SWY: No
25 (2)	253.03° GEO 254° MAG	3200 x 45	365053.3325N 0022110.5670W	THR: 21.5 m/70 ft TDZ: 21.5 m/70 ft	No	60 x 150	3320 x 300	Si/Yes	190 x 150	RWY: Asfalto/Asphalt PCN 59/F/B/W/T SWY: No

Figura C.1 Características físicas de la pista. [12]

Se definen, por lo tanto, los siguientes parámetros:

- Longitud de pista: 3200m
- Tipo de aproximación: Precisión CAT I
- Categoría de la pista: D
- Posición del umbral desplazado: 0m
- OCA: 150m
- Punto de referencia de la pista (ARP): 36º50’38’’N 002º22’12’’W
- Cabeceras:
    - Despegue 74º
    - Aterrizaje 254º
- Zona libre de obstáculos (cwy): 60m

- Elevación (z): 10m
- Escala del plano: 1:1

Instalación (VAR) Facility (VAR)	ID	FREQ	HR	Coordenadas Coordinates	ELEV DME
DME	AMR	CH 88X	H24	364959.1N 0021534.0W	60 m / 196 ft
LOC 25 (1°W) ILS CAT I	IAM	109.900 MHz	H24	365020.6N 0022324.0W	
GP 25		333.800 MHz	H24	365045.9N 0022123.7W	

Figura C.2 Instalaciones radioeléctricas que se introducirán como parámetros. [12]

Debido a que las coordenadas no nos las proporcionan en formato UTM, es necesario pasarlos al formato correcto.

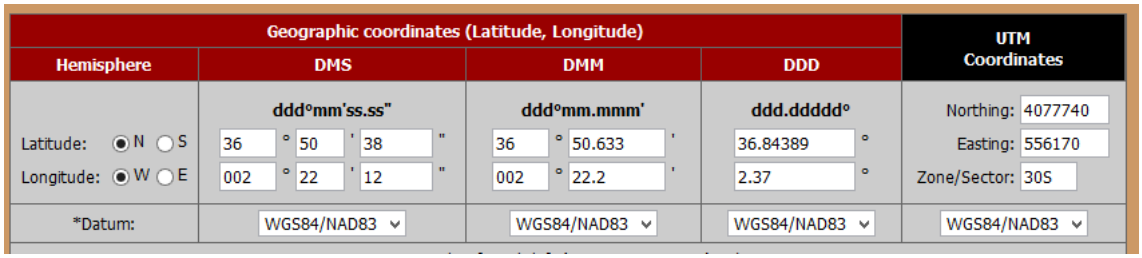

Figura C.3 Convertidor de coordenadas. [13]

Tabla C.1. Conversión a coordenadas UTM. [12] [14]

Instalación	Coordenadas WGS84	Coordenadas UTM
ARP	36º 50’ 38’’ N	X: 4077740m
	002º 22’ 12’’ W	Y: 556170m
AMR	36º 49’ 59.1’’ N	X: 4076612m
	002º 15’ 34’’ W	Y: 566036m
IAM-LOC 25	36º 50’ 20.6’’ N	X: 4088815m
	002º 23’ 24’’ W	Y: 554204m

Después de haber introducido los anteriores valores, se procede a introducir los siguientes objetos. Se introducirán 2 veces, la primera con una elevación *z* muy grande para asegurar de que existirá intersección y la segunda vez con una elevación igual a 0m, con lo cual, al no cortar con las superficies no existiría intersección.

Tabla C.2. Lista de objetos a analizar.

Identificador	X (m)	Y (m)	Z (m)
1	4076110	555888	1000
			0
2	4079284	556805	1000
			0
3	4074250	555515	1000
			0
4	4084515	558065	1000
			0
5	4078990	556355	1000
			0
6	4072890	556184	1000
			0
7	4075500	555975	1000
			0
8	4088888	554222	1000
			0
9	4090255	554630	1000
			0

En las siguientes figuras, se pueden apreciar el emplazamiento de los diferentes objetos.

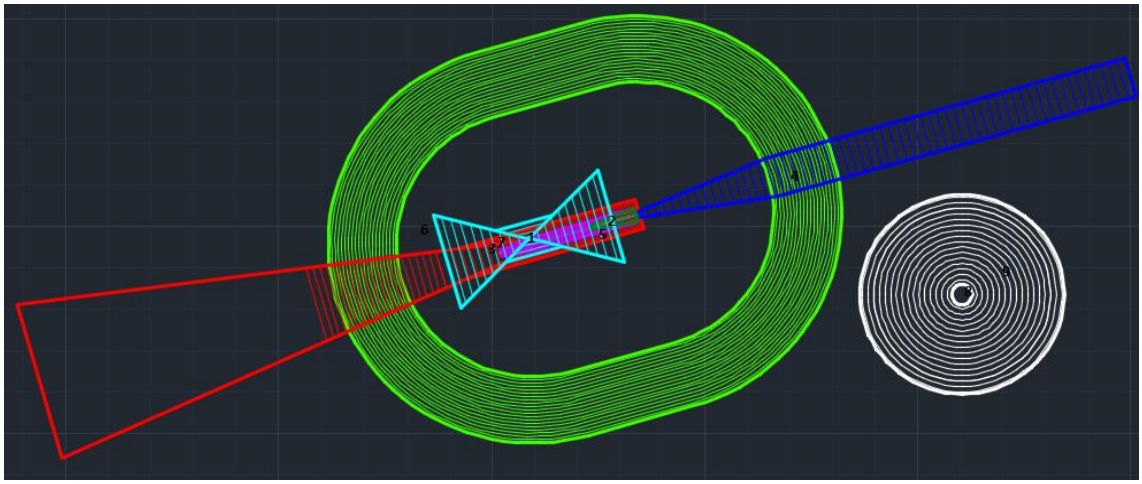

Figura C.4 Situación global de los objetos.

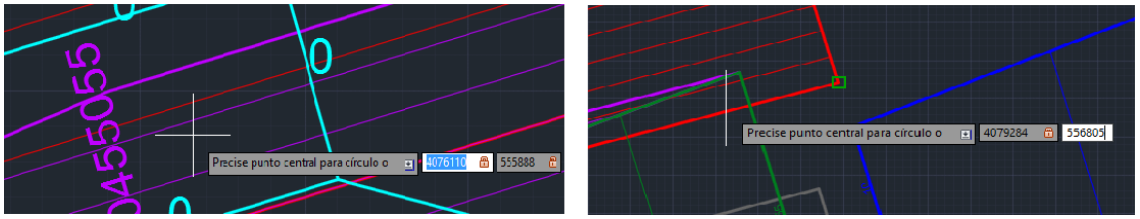

Figura C.5 Emplazamiento de los objetos 1(izquierda) y 2 (derecha).

En caso de intersección, el primer objeto intersecta con las superficies:

- Horizontal interna a una elevación de 45m.
- Transición a una elevación de 13.52m.
- Transición interna a una elevación de 49.36m.
- Zona de seguridad de la superficie del ILS/LOC a una elevación de 9m.

En caso de intersección, el segundo objeto intersecta con las superficies:

- Horizontal interna a una elevación de 45m.
- Transición a una elevación de 14,98.

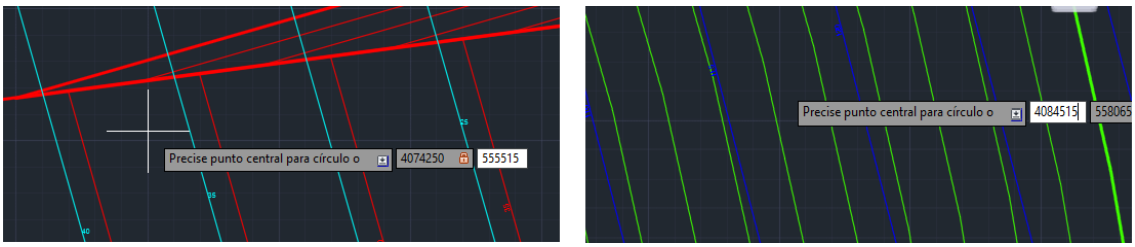

Figura C.6 Emplazamiento de los objetos 3 (izquierda) y 4 (derecha).

En caso de intersección, el tercer objeto intersecta con las superficies:

- Horizontal interna a una elevación de 45m.
- Aproximación a una elevación de 47.51m.
- Zona de limitación de alturas de la superficie del ILS/LOC a una elevación de 36.95m.

En caso de intersección, el cuarto objeto intersecta con las superficies:

- Cónica a una elevación de 116.75m.
- Ascenso en el despegue a una elevación de 117.81.

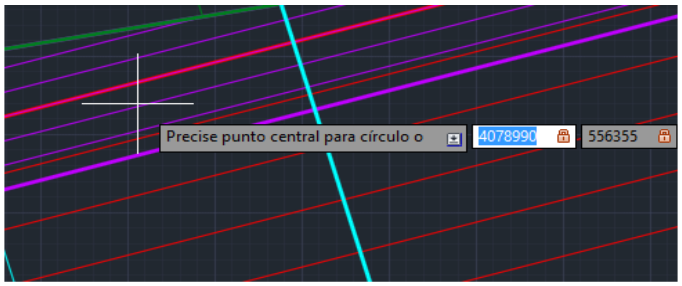

Figura C.7 Emplazamiento del objeto 5.

En caso de intersección, el quinto objeto intersecta con las superficies:

- Horizontal interna a una elevación de 45m.
- Transición a una elevación de 11.6m.
- Transición interna a una elevación de 44.2m.

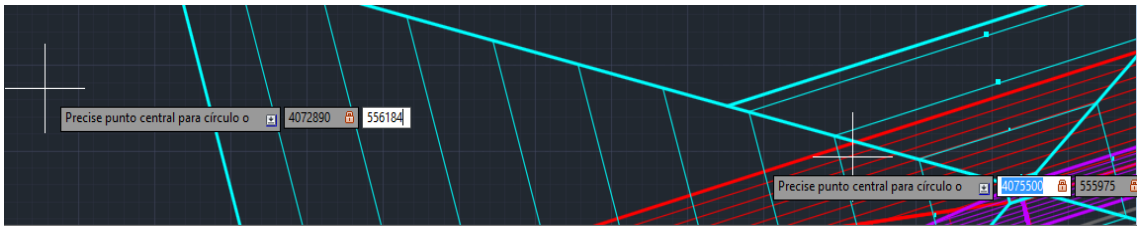

Figura C.8 Emplazamiento de los objetos 6 (izquierda) y 7 (derecha).

En caso de intersección, el sexto objetointersecta con la superficie:

- Horizontal interna a una elevación de 45m.

En caso de intersección, el séptimo objeto intersecta con la superficie:

- Transición a una elevación de 50.04m.
- Zona de limitación de alturas de la superficie del ILS/LOC a una elevación de 10.02m.

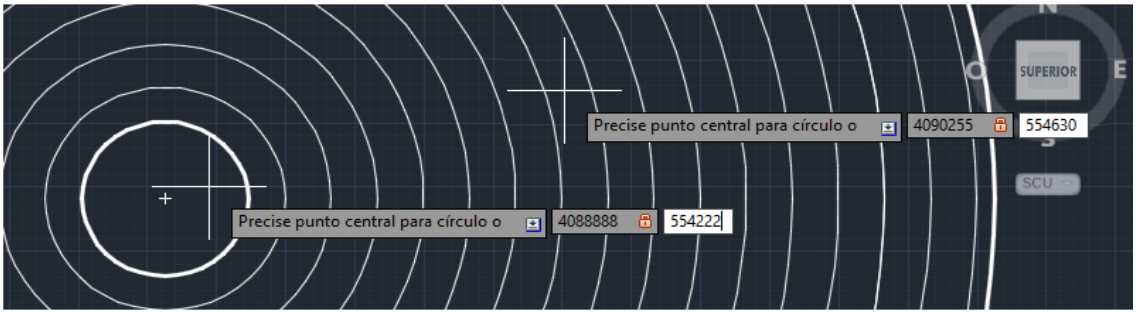

Figura C.9 Emplazamiento de los objetos 8 (izquierda) y 9 (derecha).

En caso de intersección, el octavo objeto intersecta con:

- Zona de seguridad de la servidumbre radioeléctrica a una elevación de 66m.

En caso de intersección, el noveno objeto intersecta con la superficie:

- Zona de limitación de alturas de la servidumbre radioeléctrica a una elevación de 102.3m.

En la siguiente figura, se puede apreciar el diseño que se ha obtenido mediante el aplicativo, y el diseño real de las servidumbres aeronáuticas del aeropuerto de Almería. [15]

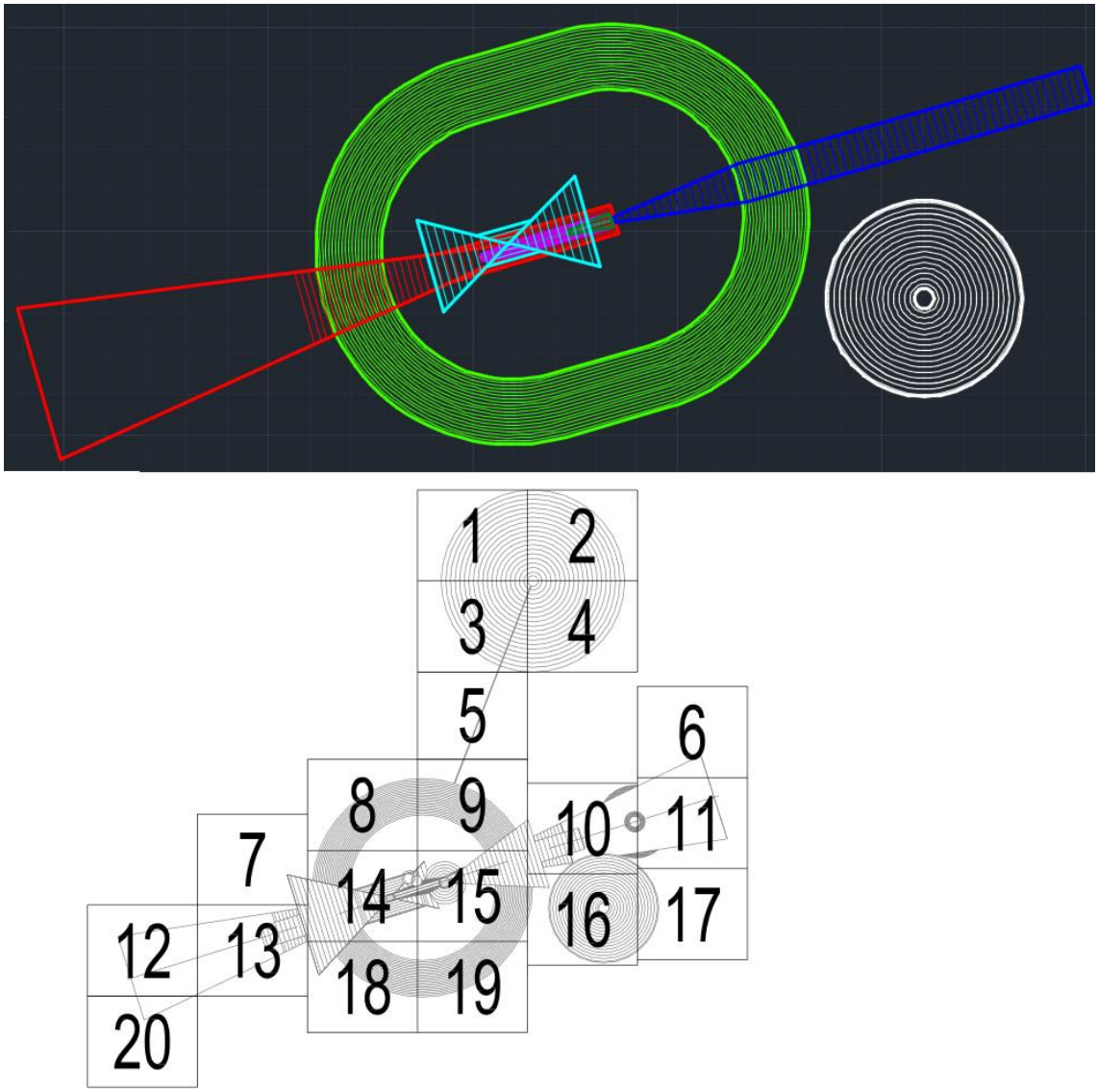

Figura C.10 Diseño real (abajo) y diseño simulado (arriba).

Las principales diferencias que se pueden observar son:

- La superficie cónica y horizontal del diseño real presentan una forma circular (cuadrados 8, 9, 18 y 19). Mientras que el diseño simulado, se ha optado por una forma elíptica.
- El diseño real posee más servidumbres, debido a que tiene en cuenta todas las configuraciones de pista de aterrizaje y despegue. Mientras que el diseño simulado, solo ha tenido en cuenta una configuración. Por ejemplo, en la casilla 11 se encuentra la superficie de aproximación de la cabecera 07, esta superficie no está representada en el diseño simulado.
- De la misma manera, en el diseño simulado solo se han representado 2 servidumbres radioeléctricas (VOR AMR e IAM-LOC 25). Mientras que en el diseño real, están representadas las servidumbres de todos las instalaciones radioeléctricas (radares, radiogoniómetros, DME, NBD, ILS-GP).
- Es importante destacar, que el diseño real se ha realizado mediante la normativa especificada en el Real Decreto 584/1972 de 24 de Febrero de la BOE [2]. Mientras que en el diseño simulado, las servidumbres físicas se han definido mediante la normativa del Anexo 14 de la OACI [1]. Consecuentemente, pueden existir pequeñas variaciones entre los valores de la elevación, pendientes, divergencias, longitudes.

Por lo tanto, a continuación se introducen los datos de la otra configuración de operaciones. Los parámetros de la cual, podemos apreciar el la *Figura C.11*.

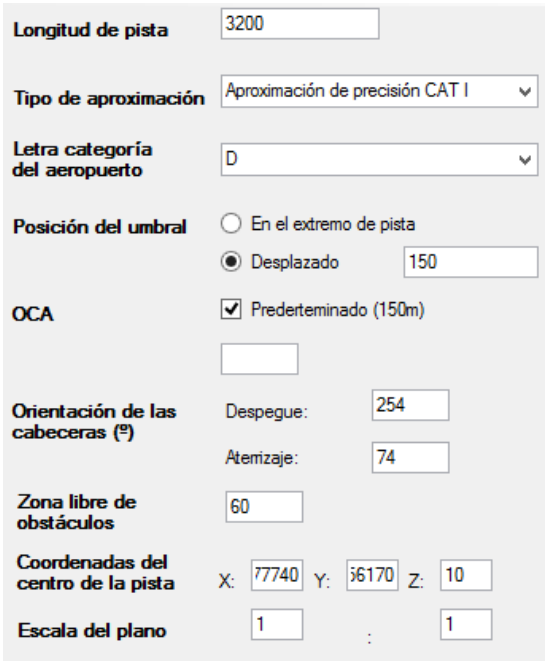

Figura C.11 Datos para la configuración: aterrizajes por cabecera 07 y despegues por cabecera 25. [12]

Se observa el siguiente resultado:

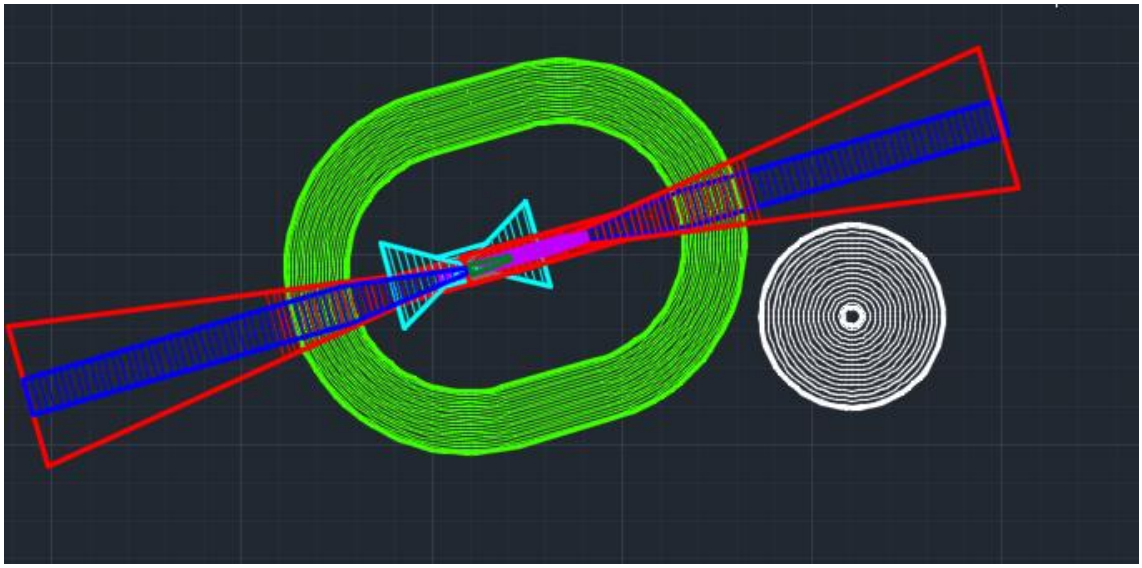

Figura C.12 Diseño simulado con las 2 configuraciones de pista.

El diseño es mucho más parecido al diseño real que el diseño simulado inicial. Las superficies cónica y horizontal interna presentarían el mismo aspecto. Por otro lado, se han representado nuevas superficies de transición, transición interna, aproximación, aproximación interna, de ascenso en el despegue y aterrizaje interrumpido.

Otro aspecto a tener en cuenta, es que las nuevas superficies se han representado encima de las anteriores razón algunas de las anteriores superficies no se pueden apreciar tan bien como antes. A diferencia del diseño real, que solo se pueden apreciar las superficies más críticas y ocultando las otras. En la *Figura C.13*, se observa que en el diseño real se puede ver como las curvas de nivel se ocultan cuando entran en contacto con la superficie de ascenso en el despegue.

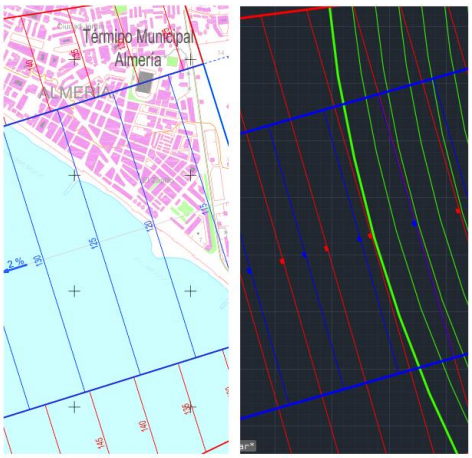

Figura C.13 Detalle de superposición de superficies en el diseño simulado y en el diseño real. [16]

A falta de representar el resto de servidumbres radioeléctricas, se puede asumir que la solución obtenida es válida ya que si se introducen los parámetros necesarios. Se conseguirían representar el resto de servidumbres que el diseño simulado no tiene definidos.

ANEXO D. Código de la aplicación

En este anexo, se adjuntará el código de la aplicación.

Class1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LibreriaDimPen
{
    //En esta clase definiremos la estructura del constructor que contendrá todos
los datos de las dimesiones y las pendientes que deben tener las
    //superfícies limitadoras de obstáculos según la tabla 4.1 del ANEXO 14 de la
OACI, también definiremos los métodos gets a partir de los cuales
    //podremos obtener los valores almacenados en ésta estructura.
    public class DimPen
    {
        double PenCon;
        double AltCon;
        double AltHint;
        double RadHint;
        double AnchApint;
        double DistUAint;
        double LongAint;
        double PenAint;
        double LongBIAp;
        double DistUAp;
        double DivAp;
        double PSecLong;
        double PSecPen;
        double SsecAlt;
        double SSecLong;
        double SSecPen;
        double SHorLong;
        double SHorLongT;
        double PenTrans;
        double PenTransInt;
        double LongBISAI;
        double DistUSAI;
        double DivSAI;
        double PenSAI;
        double LongbiASCD;
        double DistextpASCD;
        double DivASCD;
        double AnchASCD;
        double LongASCD;
        double PenASCD;

        //Constructor vacío

        public DimPen()
        {
            this.PenCon = 0;
            this.AltCon = 0;
            this.AltHint = 0;
```

```
            this.RadHint = 0;
            this.AnchApint = 0;
            this.DistUAint = 0;
            this.LongAint = 0;
            this.PenAint = 0;
            this.LongBIAp = 0;
            this.DistUAp = 0;
            this.DivAp = 0;
            this.PSecLong = 0;
            this.PSecPen = 0;
            this.SsecAlt = 0;
            this.SSecLong = 0;
            this.SSecPen = 0;
            this.SHorLong = 0;
            this.SHorLongT = 0;
            this.PenTrans = 0;
            this.PenTransInt = 0;
            this.LongBISAI = 0;
            this.DistUSAI = 0;
            this.DivSAI = 0;
            this.PenSAI = 0;
            this.LongbiASCD = 0;
            this.DistextpASCD = 0;
            this.DivASCD = 0;
            this.AnchASCD = 0;
            this.LongASCD = 0;
            this.PenASCD = 0;
        }

        //Constructor Rellenado

        public DimPen (double pencon, double altcon, double althint, double
radhint,
        double anchapint, double distuaint,double longaint, double penaint,
double longbiap,
        double distuap,double divap, double pseclong, double psecpen, double
ssecalt, double sseclong,
        double ssecpen,double shorlong,double shorlongt, double pentrans, double
pentransint,
        double longbisai, double distusai, double divsai, double pensai, double
longbiascd, double distextpascd,
        double divascd, double anchascd, double longascd, double penascd)
        {
            this.PenCon = pencon;
            this.AltCon = altcon;
            this.AltHint = althint;
            this.RadHint = radhint;
            this.AnchApint = anchapint;
            this.DistUAint = distuaint;
            this.LongAint = longaint;
            this.PenAint = penaint;
            this.LongBIAp = longbiap;
            this.DistUAp = distuap;
            this.DivAp = divap;
            this.PSecLong = pseclong;
            this.PSecPen = psecpen;
            this.SsecAlt = ssecalt;
            this.SSecLong = sseclong;
            this.SSecPen = ssecpen;
            this.SHorLong = shorlong;
            this.SHorLongT = shorlongt;
            this.PenTrans = pentrans;
```

```
            this.PenTransInt = pentransint;
            this.LongBISAI = longbisai;
            this.DistUSAI = distusai;
            this.DivSAI = divsai;
            this.PenSAI = pensai;
            this.LongbiASCD = longbiascd;
            this.DistextpASCD = distextpascd;
            this.DivASCD = divascd;
            this.AnchASCD = anchascd;
            this.LongASCD = longascd;
            this.PenASCD = penascd;
        }

        //Métodos que nos devuelven las caracteristicas dimensionales de la
servidumbre
        public double Getpencon()
        {
            return this.PenCon;
        }
        public double Getaltcon()
        {
            return this.AltCon;
        }
        public double Getalthint()
        {
            return this.AltHint;
        }
        public double Getradhint()
        {
            return this.RadHint;
        }
        public double Getanchapint()
        {
            return this.AnchApint;
        }
        public double Getdistuaint()
        {
            return this.DistUAint;
        }
        public double Getlongaint()
        {
            return this.LongAint;
        }
        public double Getpenaint()
        {
            return this.PenAint;
        }
        public double Getlongbiap()
        {
            return this.LongBIAp;
        }
        public double Getdistuap()
        {
            return this.DistUAp;
        }
        public double Getdivap()
        {
            return this.DivAp;
        }
        public double Getpseclong()
        {
            return this.PSecLong;
```

```
}
public double Getpsecpen()
{
    return this.PSecPen;
}
public double Getsesecalt()
{
    return this.SsecAlt;
}
public double Getsseclong()
{
    return this.SSecLong;
}
public double Getssecpen()
{
    return this.SSecPen;
}
public double Getshorlong()
{
    return this.SHorLong;
}
public double Getshorlongt()
{
    return this.SHorLongT;
}
public double Getpentrans()
{
    return this.PenTrans;
}
public double Getpentransint()
{
    return this.PenTransInt;
}
public double Getlongbisai()
{
    return this.LongBISAI;
}
public double Getdistusai()
{
    return this.DistUSAI;
}
public double Getdivsai()
{
    return this.DivSAI;
}
public double Getpensai()
{
    return this.PenSAI;
}
public double Getlongbiascd()
{
    return this.LongbiASCD;
}
public double Getdistextpascd()
{
    return this.DistextpASCD;
}
public double Getdivascd()
{
    return this.DivASCD;
}
public double Getanchascd()
```

```
        {
            return this.AnchASCD;
        }
        public double Getlongascd()
        {
            return this.LongASCD;
        }
        public double Getpenascd()
        {
            return this.PenASCD;
        }
        public double Getssecalt()
        {
            return this.SsecAlt;
        }
        // Método para definir los valores que deberán tener las diferentes
superfícies dependiendo de la longitud de pista y el tipo de
        //aproximación que siga.
        public DimPen GetDimPen(double longitudpista, string aprox, string
catletra, double OCA, Franjadim fjd, double cwy, bool noche, double penascd)
        {
            int clave=0;
            DimPen dp = new DimPen(this.PenCon, this.AltCon, this.AltHint,
this.RadHint, this.AnchApint, this.DistUAint,
            this.LongAint, this.PenAint, this.LongBIAp, this.DistUAp, this.DivAp,
this.PSecLong, this.PSecPen,this.SsecAlt, this.SSecLong, this.SSecPen,
            this.SHorLong, this.SHorLongT, this.PenTrans, this.PenTransInt,
this.LongBISAI, this.DistUSAI, this.DivSAI, this.PenSAI,this.LongbiASCD,
            this.DistextpASCD, this.DivASCD, this.AnchASCD, this.LongASCD,
this.PenASCD);

            if (longitudpista < 800)
            {
                clave = 1;
            }
            if (longitudpista >= 800 && longitudpista < 1200)
            {
                clave = 2;
            }
            if (longitudpista >= 1200 && longitudpista <= 1800)
            {
                clave = 3;
            }
            if (longitudpista >1800)
            {
                clave = 4;
            }
            dp.SsecAlt = 150;

            if (clave == 1)
            {
                dp.LongbiASCD = 60;
                dp.DistextpASCD = 30;
                dp.DivASCD = 10;
                dp.AnchASCD = 380;
                dp.LongASCD = 1600;
                dp.PenASCD = 5;
            }
            else
            {
                if (clave == 2)
                {
```

```
                    dp.LongbiASCD = 80;
                    dp.DistextpASCD = 60;
                    dp.DivASCD = 10;
                    dp.AnchASCD = 580;
                    dp.LongASCD = 2500;
                    dp.PenASCD = 4;
                }
                else
                {
                    dp.DistextpASCD = 60;
                    dp.DivASCD = 12.5;
                    if (noche == false)
                    {
                        dp.AnchASCD = 1200;
                    }
                    else
                    {
                        dp.AnchASCD = 1800;
                    }
                    dp.LongbiASCD = 180;
                    dp.PenASCD = penascd;
                    if (dp.PenASCD == 2)
                    {
                        dp.LongASCD = 15000;
                    }
                    else
                    {
                        dp.LongASCD = 300 * 100 / dp.PenASCD;
                    }
                }
            }
            if (cwy > dp.DistextpASCD)
            {
                dp.DistextpASCD = cwy;
            }
            if (noche == true && (aprox == "Visual" || aprox == "Noprec") &&
(clave == 3 || clave == 4))
            {
                dp.AnchASCD = 1800;
            }
            if (aprox=="Visual")
            {
                if (clave==1)
                {
                    dp.PenCon = 5;
                    dp.AltCon = 35;
                    dp.AltHint = 45;
                    dp.RadHint = 2000;
                    dp.AnchApint = 0;
                    dp.DistUAint = 0;
                    dp.LongAint = 0;
                    dp.PenAint = 0;
                    dp.LongBIAp = 60;
                    dp.DistUAp = 30;
                    dp.DivAp = 10;
                    dp.PSecLong = 1600;
                    dp.PSecPen = 5;
                    dp.SSecLong = 0;
                    dp.SSecPen = 0;
                    dp.SHorLong = 0;
                    dp.SHorLongT = 0;
                    dp.PenTrans = 20;
```

```
        dp.PenTransInt = 0;
        dp.LongBISAI = 0;
        dp.DistUSAI = 0;
        dp.DivSAI = 0;
        dp.PenSAI = 0;
    }
    if (clave==2)
    {
        dp.PenCon = 5;
        dp.AltCon = 55;
        dp.AltHint = 45;
        dp.RadHint = 2500;
        dp.AnchApint = 0;
        dp.DistUAint = 0;
        dp.LongAint = 0;
        dp.PenAint = 0;
        dp.LongBIAp = 80;
        dp.DistUAp = 60;
        dp.DivAp = 10;
        dp.PSecLong = 2500;
        dp.PSecPen = 4;
        dp.SSecLong = 0;
        dp.SSecPen = 0;
        dp.SHorLong = 0;
        dp.SHorLongT = 0;
        dp.PenTrans = 20;
        dp.PenTransInt = 0;
        dp.LongBISAI = 0;
        dp.DistUSAI = 0;
        dp.DivSAI = 0;
        dp.PenSAI = 0;
    }
    if (clave == 3)
    {
        dp.PenCon = 5;
        dp.AltCon = 75;
        dp.AltHint = 45;
        dp.RadHint = 4000;
        dp.AnchApint = 0;
        dp.DistUAint = 0;
        dp.LongAint = 0;
        dp.PenAint = 0;
        dp.LongBIAp = 150;
        dp.DistUAp = 60;
        dp.DivAp = 10;
        dp.PSecLong = 3000;
        dp.PSecPen = 3.33;
        dp.SSecLong = 0;
        dp.SSecPen = 0;
        dp.SHorLong = 0;
        dp.SHorLongT = 0;
        dp.PenTrans = 14.3;
        dp.PenTransInt = 0;
        dp.LongBISAI = 0;
        dp.DistUSAI = 0;
        dp.DivSAI = 0;
        dp.PenSAI = 0;
    }
    if (clave == 4)
    {
        dp.PenCon = 5;
        dp.AltCon = 100;
```

```
            dp.AltHint = 45;
            dp.RadHint = 4000;
            dp.AnchApint = 0;
            dp.DistUAint = 0;
            dp.LongAint = 0;
            dp.PenAint = 0;
            dp.LongBIAp = 150;
            dp.DistUAp = 60;
            dp.DivAp = 10;
            dp.PSecLong = 3000;
            dp.PSecPen = 2.5;
            dp.SSecLong = 0;
            dp.SSecPen = 0;
            dp.SHorLong = 0;
            dp.SHorLongT = 0;
            dp.PenTrans = 14.3;
            dp.PenTransInt = 0;
            dp.LongBISAI = 0;
            dp.DistUSAI = 0;
            dp.DivSAI = 0;
            dp.PenSAI = 0;
        }
    }
    if (aprox == "Noprec")
    {
        if (clave == 1 || clave == 2)
        {
                dp.PenCon = 5;
                dp.AltCon = 60;
                dp.AltHint = 45;
                dp.RadHint = 3500;
                dp.AnchApint = 0;
                dp.DistUAint = 0;
                dp.LongAint = 0;
                dp.PenAint = 0;
                dp.LongBIAp = 150;
                dp.DistUAp = 60;
                dp.DivAp = 15;
                dp.PSecLong = 2500;
                dp.PSecPen = 3.33;
                dp.SSecLong = 0;
                dp.SSecPen = 0;
                dp.SHorLong = 0;
                dp.SHorLongT = 0;
                dp.PenTrans = 20;
                dp.PenTransInt = 0;
                dp.LongBISAI = 0;
                dp.DistUSAI = 0;
                dp.DivSAI = 0;
                dp.PenSAI = 0;
        }
        if (clave == 3)
        {
            dp.PenCon = 5;
            dp.AltCon = 75;
            dp.AltHint = 45;
            dp.RadHint = 4000;
            dp.AnchApint = 0;
            dp.DistUAint = 0;
            dp.LongAint = 0;
            dp.PenAint = 0;
            dp.LongBIAp = 300;
```

```
            dp.DistUAp = 60;
            dp.DivAp = 15;
            dp.PSecLong = 3000;
            dp.PSecPen = 2;
            dp.SSecLong = 3600;
            dp.SSecPen = 2.5;
            dp.SHorLong = 8400;
            dp.SHorLongT = 15000;
            dp.PenTrans = 14.3;
            dp.PenTransInt = 0;
            dp.LongBISAI = 0;
            dp.DistUSAI = 0;
            dp.DivSAI = 0;
            dp.PenSAI = 0;
        }
        if (clave == 4)
        {
            dp.PenCon = 5;
            dp.AltCon = 100;
            dp.AltHint = 45;
            dp.RadHint = 4000;
            dp.AnchApint = 0;
            dp.DistUAint = 0;
            dp.LongAint = 0;
            dp.PenAint = 0;
            dp.LongBIAp = 300;
            dp.DistUAp = 60;
            dp.DivAp = 15;
            dp.PSecLong = 3000;
            dp.PSecPen = 2;
            dp.SSecLong = 3600;
            dp.SSecPen = 2.5;
            dp.SHorLong = 8400;
            dp.SHorLongT = 15000;
            dp.PenTrans = 14.3;
            dp.PenTransInt = 0;
            dp.LongBISAI = 0;
            dp.DistUSAI = 0;
            dp.DivSAI = 0;
            dp.PenSAI = 0;
        }
    }
    if (aprox == "Prec1")
    {
        if (clave == 1 || clave == 2)
        {
            dp.PenCon = 5;
            dp.AltCon = 60;
            dp.AltHint = 45;
            dp.RadHint = 3500;
            dp.AnchApint = 90;
            dp.DistUAint = 60;
            dp.LongAint = 900;
            dp.PenAint = 2.5;
            dp.LongBIAp = 150;
            dp.DistUAp = 60;
            dp.DivAp = 15;
            dp.PSecLong = 3000;
            dp.PSecPen = 2.5;
            dp.SSecLong = 12000;
            dp.SSecPen = 3;
            dp.SHorLong = 0;
```

```
            dp.SHorLongT = 15000;
            dp.PenTrans = 14.3;
            dp.PenTransInt = 40;
            dp.LongBISAI = 90;
            dp.DistUSAI = longitudpista + fjd.Getfranlong();
            dp.DivSAI = 10;
            dp.PenSAI = 4;
        }
        else
        {
                dp.PenCon = 5;
                dp.AltCon = 100;
                dp.AltHint = 45;
                dp.RadHint = 4000;
                dp.AnchApint = 120;
                dp.DistUAint = 60;
                dp.LongAint = 900;
                dp.PenAint = 2;
                dp.LongBIAp = 300;
                dp.DistUAp = 60;
                dp.DivAp = 15;
                dp.PSecLong = 3000;
                dp.PSecPen = 2;
                dp.SSecLong = 3600;
                dp.SSecPen = 2.5;
                dp.SHorLong = 8400;
                dp.SHorLongT = 15000;
                dp.PenTrans = 14.3;
                dp.PenTransInt = 33.3;
                dp.LongBISAI = 120;
                dp.DistUSAI = 1800;
                dp.DivSAI = 10;
                dp.PenSAI = 3.33;
        }
    }
    if (aprox == "Prec23")
    {
        dp.PenCon = 5;
        dp.AltCon = 100;
        dp.AltHint = 45;
        dp.RadHint = 4000;
        dp.AnchApint = 120;
        dp.DistUAint = 60;
        dp.LongAint = 900;
        dp.PenAint = 2;
        dp.LongBIAp = 300;
        dp.DistUAp = 60;
        dp.DivAp = 15;
        dp.PSecLong = 3000;
        dp.PSecPen = 2;
        dp.SSecLong = 3600;
        dp.SSecPen = 2.5;
        dp.SHorLong = 8400;
        dp.SHorLongT = 15000;
        dp.PenTrans = 14.3;
        dp.PenTransInt = 33.3;
        dp.LongBISAI = 120;
        dp.DistUSAI = 1800;
        dp.DivSAI = 10;
        dp.PenSAI = 3.33;
    }
```

```
            if ((aprox == "Prec1" || aprox == "Prec23") && (clave == 3 || clave
== 4) && catletra=="F")
            {
                dp.AnchApint = 155;
                dp.LongBISAI = 155;
            }
            if (OCA != 150 && (clave == 3 || clave == 4) && (aprox == "Noprec" ||
aprox == "Prec1" || aprox == "Prec23"))
            {
                dp.SsecAlt = OCA;
                dp.SSecLong = (OCA * 100 - dp.PSecLong * dp.PSecPen) /
dp.SSecPen;
                dp.SHorLong = dp.SHorLongT - dp.PSecLong - dp.SSecLong;
            }
            return dp;
        }
    }

    //Clase que contendrá los valores dimensionales de la franja
    public class Franjadim
    {
        double longitud;
        double ancho;

        public Franjadim()
        {
            this.longitud = 0;
            this.ancho = 0;
        }

        public Franjadim(double longitud, double ancho)
        {
            this.longitud = longitud;
            this.ancho = ancho;
        }

        public Franjadim GetFranjadim(string aprox, double longitudpista)
        {
            int clave=0;
            Franjadim fjd = new Franjadim(this.longitud, this.ancho);
            if (longitudpista < 800)
            {
                clave = 1;
            }
            if (longitudpista >= 800 && longitudpista < 1200)
            {
                clave = 2;
            }
            if (longitudpista >= 1200 && longitudpista <= 1800)
            {
                clave = 3;
            }
            if (longitudpista > 1800)
            {
                clave = 4;
            }
            if(clave==2 || clave==3 || clave==4 || (aprox!="Visual" && clave==1))
            {
                fjd.longitud=60;
            }
            if (aprox == "Visual")
            {
```

```
                if (clave == 1)
                {
                    fjd.longitud = 30;
                    fjd.ancho = 30;
                }
                if (clave == 2)
                {
                    fjd.ancho=40;
                }
                if(clave==3 || clave==4)
                {
                    fjd.ancho=75;
                }
            }
            else
            {
                if(clave==1 || clave==2)
                {
                    fjd.ancho=75;
                }
                else
                {
                    fjd.ancho=150;
                }
            }
            return fjd;
        }

        //Métodos GET
        public double Getfranlong()
        {
            return this.longitud;
        }
        public double Getfrananch()
        {
            return this.ancho;
        }
        //Métodos SET
        public void Setfranlong(double frl)
        {
            this.longitud = frl;
        }
        public void Setfrananch(double fra)
        {
            this.ancho = fra;
        }
    }

    //Clase que contiene los valores de las superficies radioeléctricas excepto
ILS
    public class SRad
    {
        double zseg;
        double zalt;
        double pen;
        string id;
        string tipo;
        string ant;
        double[] radpos = new double[3];
        public SRad()
        {
            this.zseg = 0;
```

```
            this.zalt = 0;
            this.pen = 0;
            this.id = null;
            this.tipo = null;
            this.ant = null;
            this.radpos[0] = 0;
            this.radpos[1] = 0;
            this.radpos[2] = 0;
        }

        public SRad(double zseg, double zalt, double pen, string id, string tipo,
string ant, double [] radpos)
        {
            this.zseg = zseg;
            this.zalt = zalt;
            this.pen = pen;
            this.id = id;
            this.tipo = tipo;
            this.ant = ant;
            this.radpos = radpos;
        }

        //MÉTODOS GETS

        //Necesitará 2 variables de tipo string en los cuales se indicará si la
instalación radioeléctrica es de
        //tipo "centro de comunicaciones" o "ayuda radioeléctrica", la segunda
variable indicará qué tipo de "centro" o "ayuda" es.
        //Las otras 2 variables d etipo double son para el caso en que tengamos
un centro de comunicación que actue como enlace hertziano
        //en ese caso es necesario esta información para obtener el valor de la
pendiente y de la zona de limitación de alturas.
        public SRad GetSRad(string tipo, string ant, string id, double [] radpos)
        {
            SRad srad = new SRad();
            srad.id = id;
            srad.radpos[0] = radpos[0];
            srad.radpos[1] = radpos[1];
            srad.radpos[2] = radpos[2];
            if (tipo == "cc")
            {
                srad.tipo = "Centro de comunicaciones.";
                srad.zalt = 2000;
                if (ant == "Frecuencias bajas (LF) o medias (MF)")
                {
                    srad.ant = "Frecuencias bajas (LF) o medias (MF).";
                    srad.zseg = 200;
                    srad.pen = 10;
                }
                else
                {
                    srad.zseg = 300;
                    if (ant == "Frecuencias altas (HF)")
                    {
                        srad.ant = "Frecuencias altas (HF).";
                        srad.pen = 7.5;
                    }
                    else
                    {
                        srad.ant = "Frecuencias muy altas (VHF) o ultra elevada
(UHF).";
                        srad.pen = 5;
```

```
                    }
                }

            }
            else
            {
                srad.tipo = "Ayudas a la navegación.";
                if (ant != "ILS/LOC" && ant != "ILS/GP")
                {
                    if (ant == "Radiobaliza marcadora de tipo 'Z' (75MHz)" || ant
== "Radiobaliza marcadora en abanico ('Fan Marker') (75MHz)")
                    {
                        srad.zseg = 200;
                        srad.zalt = 1000;
                        srad.pen = 100;
                        if (ant == "Radiobaliza marcadora de tipo 'Z' (75MHz)")
                        {
                            srad.ant = "Radiobaliza marcadora tipo 'Z' (75MHz).";
                        }
                        else
                        {
                            srad.ant = "Radiobaliza marcadora en abanico ('Fan
Maker') (75MHz).";
                        }
                    }
                    else
                    {
                        srad.zseg = 300;
                        if (ant == "Radiofaros no direccionales")
                        {
                            srad.ant = "Radiofaros no direccionales.";
                            srad.zalt = 2000;
                            srad.pen = 10;
                        }
                        else
                        {
                            if (ant == "Radiofaro omnidireccional VFH (VOR),
equipo medidor de distancia (DME) o TACAN")
                            {
                                srad.ant = "Radiofaro omnireccional VFH (VOR),
equipo medidor de distancia (DME) o TACAN.";
                                srad.zalt = 3000;
                                srad.pen = 3;
                            }
                            else
                            {
                                srad.zalt = 5000;
                                srad.pen = 2;
                                if (ant == "Radar de vigilancia primario o
secundario (SSR)")
                                {
                                    srad.ant = "Radar de vigilangia primario o
secundario (SSR).";
                                }
                                else
                                {
                                    srad.ant = "Radiogoniómetro VHF (VDF) o UHF
(UDF).";
                                }
                            }
                        }
                    }
```

```
                }
                else
                {
                    srad.pen = 2;
                    srad.zseg = 0;
                    srad.zalt = 0;
                    if (ant == "ILS/LOC")
                    {
                        srad.ant = "ILS/LOC";
                    }
                    else
                    {
                        srad.ant = "ILS/GP";
                    }
                }
            }
            return srad;
        }

        public double Getzseg()
        {
            return this.zseg;
        }
        public double Getzalt()
        {
            return this.zalt;
        }
        public double Getpen()
        {
            return this.pen;
        }
        public string Getid()
        {
            return this.id;
        }
        public string Gettipo()
        {
            return this.tipo;
        }
        public string Getant()
        {
            return this.ant;
        }
        public double [] Getradpos()
        {
            return this.radpos;
        }
    }

    //Clase que la identificación y las coordenadas de los obstáculos
    public class OBS
    {
        string id;
        double[] pos = new double[3];
        //Constructor vacío
        public OBS()
        {
            this.id = null;
            this.pos[0] = 0;
            this.pos[1] = 0;
            this.pos[2] = 0;
        }
```

```
        //Constructor rellenado
        public OBS(string id, double[] pos)
        {
            this.id = id;
            this.pos[0] = pos[0];
            this.pos[1] = pos[1];
            this.pos[2] = pos[2];
        }
        //Método SET
        public void Setinfo(string id, double[] pos)
        {
            this.id = id;
            this.pos[0] = pos[0];
            this.pos[1] = pos[1];
            this.pos[2] = pos[2];
        }
        //Métodos Gets
        public string Getid()
        {
            return this.id;
        }
        public double[] Getpos()
        {
            return this.pos;
        }
    }
}
```

Dibuclass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using LibreriaDimPen;
using LibreriaIntersecciones;


namespace DibujarSuperficies
{
    public class Dibuclass
    {
        DimPen dp = new DimPen();
        Intersecciones its = new Intersecciones();
        Franjadim fjd = new Franjadim();
        //Variable de la escala del mapa (si se va a cargar uno) de fondo, donde
estarán representadas las superficies limitadoras de obstáculos.
        double[] escala = { 1, 1 };
        //Variable en la cual estará guardado la coordenada de la esquina
inferior del mapa( en caso de que se haya cargado uno).
        double[] mapapos = { 0, 0 };
        //Variable que contiene la orientación de la pista respecto el norte.
        double ang;
        //Variable que indica el tipo de aproximación
        string aprox;
        double longitudpista;
        //Variable de una posición de intersección imaginaria (nos aseguraremos
de que no intersecte, lo utilizaremos para la obtención de los puntos necesarios
        //para representar las superfícies limitadoras de obstáculos.
        double[] pos = { 99999999999999999, 99999999999999999,
9999999999999999999 };
       //Variable con la posición de la pista
        double [] pistapos;
        //Variable temporal con el valor de la pista desplazado
        double[] temppistapos;
        //Variables que indicarán el sentido la cabecera de aterrizaje y de
despegue
        double sentaterrizaje;
        double sentdespegue;
        //variable que indicará los m que se ha desplazado el umbral
        double posumbral;
        //Variable que indicará la clave de la pista
        int clave;
        //Matriz que contendrá las alturas de las curvas de nivel y las
posiciones donde deseemos que se localicen los letreros que indicarán la
altura(z(m))
        //de las curvas de nivel
        double[,] zs;
        //Vector temporal que contendrá las alturas de las curvas de nivel.
        double[] tempzs = new double[999];
        //Clase que contendrá los datos de la instalación radioeléctrica
        SRad srad = new SRad();
        //Posición de la superfície radioeléctrica trasladada a un plano
horizontal
        double[] radpos;
```

```
        //Matrices con las posiciones de los letreros y sus alturas(z)
        double[,] templet1 = new double[100, 3];
        double[,] templet2 = new double[100, 3];
        double[,] templet3 = new double[100, 3];
        double[,] templet4 = new double[100, 3];
        //Posiciones de cada matriz
        int t1=0;
        int t2=0;
        //Ancho de pista
        double ancho;
        public Dibuclass()
        {
            this.dp = null;
            this.its = null;
            this.escala[0] = 1;
            this.escala[1] = 1;
            this.mapapos[0] = 0;
            this.mapapos[1] = 0;
            this.ang = 0;
            this.aprox = null;
            this.longitudpista = 0;
            this.pistapos = null;
            this.temppistapos = null;
        }
        public void Setinfo(DimPen dp, Intersecciones its, double[] escala,
double ang, string aprox,
            double longitudpista, double [] pistapos, double sentaterrizaje,
double sentdespegue,double posumbral, int clave, Franjadim fjd,double ancho)
        {
            this.dp = dp;
            this.its = its;
            this.escala = escala;
            this.ang = ang;
            this.aprox = aprox;
            this.longitudpista = longitudpista;
            this.pistapos = pistapos;
            temppistapos = new double[3];
            this.temppistapos[0] = this.pistapos[0];
            this.temppistapos[1] = this.pistapos[1];
            this.temppistapos[2] = this.pistapos[2];
            //Trasladamos las coordenadas para tener unas superfícies paralelas
al eje horizontal.
            this.temppistapos = its.giro(this.temppistapos, this.ang);
            this.sentaterrizaje = sentaterrizaje;
            this.sentdespegue = sentdespegue;
            this.posumbral = posumbral;
            this.clave = clave;
            this.fjd = fjd;
            this.ancho = ancho;
        }
        //Puntos de la pista
        public double[,] Getpista()
        {
            double[,] pista = new double[4, 2];
            pista[0, 0] = temppistapos[0] - longitudpista / 2;
            pista[0, 1] = temppistapos[1] + this.ancho / 2;
            pista[1, 0] = temppistapos[0] + longitudpista / 2;
            pista[1, 1] = temppistapos[1] + this.ancho / 2;
            pista[2, 0] = temppistapos[0] + longitudpista / 2;
            pista[2, 1] = temppistapos[1] - this.ancho / 2;
            pista[3, 0] = temppistapos[0] - longitudpista / 2;
            pista[3, 1] = temppistapos[1] - this.ancho / 2;
```

```
            //Trasladamos a la orientación inicial
            for (int i = 0; i < 4; i++)
            {
                double[] girpista = new double[2];
                girpista = its.giroinv(pista[i, 0], pista[i, 1], ang);
                pista[i, 0] = (girpista[0]) * this.escala[0] / this.escala[1];
                pista[i, 1] = (girpista[1]) * this.escala[0] / this.escala[1];
            }
            return pista;
        }
        //Métodos a partir de los cuales obtendremos los puntos necesarios para
poder representar las superficies limitadoras de obstáculos en autocad.
        //Horizontal interna.
        public double [,] GetHIp()
        {
            //DimPen dp,Intersecciones its, double ang,double longitudpista,
double [] temppistapos
            //En primer lugar definiremos la variable que contendrá todos los
puntos, en este caso nos interesará obtener los puntos que conformarán los 2
arcos
            //que tiene la horizontal interna.
            double [,] HIp=new double[6,2];
            //A partir de esta ejecución, a pesar de que no vayamos a utilizar el
valor del booleano, ejecutaremos este método para obtener los límites del
            //intervalo que conforma la HI.
            bool v = its.inthoint(this.longitudpista, dp.Getradhint(), this.pos,
temppistapos);
            //Guardamos los límites en un vector temporal.
            double[] HIplim = new double[4];
            HIplim = its.Getlim();
            //Empezamos a rellenar la matriz con los puntos, es importante tener
en cuenta que hay que volver a trasladar los puntos a la orientación original.
            HIp[0, 0] = HIplim[0];
            HIp[0, 1] = (HIplim[2] + HIplim[3]) / 2;
            HIp[1, 0] = HIplim[0] - dp.Getradhint();
            HIp[1, 1] = HIplim[2];
            HIp[2, 0] = HIplim[0] - dp.Getradhint();
            HIp[2, 1] = HIplim[3];
            HIp[3, 0] = HIplim[1];
            HIp[3, 1] = (HIplim[2] + HIplim[3]) / 2;
            HIp[4, 0] = HIplim[1] + dp.Getradhint();
            HIp[4, 1] = HIplim[2];
            HIp[5, 0] = HIplim[1] + dp.Getradhint();
            HIp[5, 1] = HIplim[3];
            //Trasladamos las coordenadas a la orientación original
            for (int i = 0; i < 6; i++)
            {
                double[] girHI = new double[2];
                girHI = its.giroinv(HIp[i, 0], HIp[i, 1], ang);
                HIp[i, 0] = (girHI[0]) * this.escala[0] / this.escala[1];
                HIp[i, 1] = (girHI[1]) * this.escala[0] / this.escala[1];
            }
            return HIp;
        }

        //CÖNICA
        public double[,] Getcon()
        {
            //Matrices temporales para los puntos del contorno y los letreros de
las profundidades
            double[,] CONtemp = new double[200, 2];
            double[,] zstemp = new double[200, 3];
```

```
            bool v = its.intconi(this.longitudpista, dp.Getradhint(), this.pos,
temppistapos, dp.Getpencon(), dp.Getaltcon());
            //Guardamos los límites en un valor temporal.
            double[] CONlim = new double[4];
            CONlim = its.Getlim();
            //Empezamos a rellenas la matriz con los puntos, los devolvemos a su
orientación original y también tendremos en cuenta el origen y la escala, en caso
de que se
            //vaya a cargar un mapa;
            double radiomax = dp.Getradhint() + (dp.Getaltcon() / (dp.Getpencon()
/ 100));
            CONtemp[0, 0] = CONlim[0];
            CONtemp[0, 1] = (CONlim[2] + CONlim[3]) / 2;
            CONtemp[1, 0] = CONlim[0] - radiomax;
            CONtemp[1, 1] = CONlim[2];
            CONtemp[2, 0] = CONlim[0] - radiomax;
            CONtemp[2, 1] = CONlim[3];
            CONtemp[3, 0] = CONlim[1];
            CONtemp[3, 1] = (CONlim[2] + CONlim[3]) / 2;
            CONtemp[4, 0] = CONlim[1] + radiomax;
            CONtemp[4, 1] = CONlim[2];
            CONtemp[5, 0] = CONlim[1] + radiomax;
            CONtemp[5, 1] = CONlim[3];
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(temppistapos[2]) + 45;
            int i = 6;
            int t = 1;
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            //Obtenemos los puntos de la horizontal interna para pintar a partir
de ellos las curvas de nivel
            double[,] HI = new double[6, 2];
            HI = GetHIp();
            //Trasladamos los puntos de la matriz a de manera que sea
horizontales
            for (int j = 0; j < 6; j++)
            {
                double[] girHI = new double[2];
                double[] HHi = { HI[j, 0], HI[j, 1], 0 };
                girHI = its.giro(HHi, ang);
                HI[j, 0] = (girHI[0]) * this.escala[0] / this.escala[1];
                HI[j, 1] = (girHI[1]) * this.escala[0] / this.escala[1];
            }
            for (double tempz = this.tempzs[1]; tempz < dp.Getaltcon() +
temppistapos[2] +45; tempz = tempz + 5)
            {
                this.tempzs[t] = tempz;
```

```
                CONtemp[i, 0] = HI[0, 0] + (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 1] = HI[0, 1];
                i = i + 1;
                CONtemp[i, 1] = HI[1, 1] + (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 0] = HI[1, 0];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = temppistapos[0];
                zstemp[t, 2] = CONtemp[i, 1];
                i = i + 1;
                CONtemp[i, 1] = HI[2, 1] - (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 0] = HI[2, 0];
                i = i + 1;
                CONtemp[i, 0] = HI[3, 0] - (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 1] = HI[3, 1];
                i = i + 1;
                CONtemp[i, 1] = HI[4, 1] + (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 0] = HI[4, 0];
                i = i + 1;
                CONtemp[i, 1] = HI[5, 1] - (this.tempzs[t] - this.tempzs[0]) *
(100 / dp.Getpencon());
                CONtemp[i, 0] = HI[5, 0];
                i = i + 1;
                t = t + 1;
            }
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = temppistapos[0];
            zstemp[0, 2] = HI[1, 1];
            this.tempzs[t] = dp.Getaltcon() + temppistapos[2] + 45;
            zstemp[t, 0] = this.tempzs[t];
            zstemp[t, 1] = temppistapos[0];
            zstemp[t, 2] = CONtemp[1, 1];
            //MAtriz que contendrá los puntos base de la CÖNICA.
            double[,] CON = new double[i, 2];
            //Transladamos las coordenadas a la orientación original
            for (int j = 0; j < i; j++)
            {
                double[] girCON = new double[2];
                girCON = its.giroinv(CONtemp[j, 0], CONtemp[j, 1], ang);
                CON[j, 0] = (girCON[0]) * this.escala[0] / this.escala[1];
                CON[j, 1] = (girCON[1]) * this.escala[0] / this.escala[1];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t + 1, 3];
            for (int j = 0; j < t + 1; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
                this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return CON;
        }

        //APROXIMACIÓN
        public double[,] Getap()
        {
```

```
            //Matrices temporales para los puntos del contorno y los letreros de
las profundidades
            double[,] APtemp = new double[200, 2];
            double[,] zstemp = new double[200, 3];
            bool[] v = its.intaprox(this.aprox, dp, this.pos, temppistapos,
this.clave, this.longitudpista, this.posumbral, this.sentaterrizaje);
            //Guardamos los límites en un valor temporal.
            double[] APlim = new double[4];
            APlim = its.Getlim();
            APtemp[0, 1] = this.temppistapos[1] + dp.Getlongbiap() / 2;
            APtemp[1, 1] = this.temppistapos[1] - dp.Getlongbiap() / 2;
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(temppistapos[2]);
            int i = 4;
            int t = 1;
            //Varibale que indicará la posición de la prufundidad dentro del
vector tempszs cuando se produzca el cambio de pendiente
            int tt = t;
            //Variable que indicará el punto donde cambia de la primera sección a
la segunda
            int a = i;
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            if (this.sentaterrizaje == -1)
            {
                APtemp[0, 0] = APlim[0];
                APtemp[1, 0] = APlim[0];
            }
            else
            {
                APtemp[0, 0] = APlim[1];
                APtemp[1, 0] = APlim[1];
            }
            //En primer lugar determinaremos si la pista es de precisión o
Instrumental con clave 3 o 4, si es el caso
            //la superficie de APROXIMACIÓN tendrá una sección secundaria y otra
sección horizontal.
            if (aprox == "Prec1" || aprox == "Prec23" || (aprox == "NoPrec" &&
(clave == 3 || clave == 4)))
            {
                //En este caso será necesario identificar los límites secundarios
de la sección secundaria y horizonta
                double[] APlim2 = new double[4];
                APlim2 = its.Getlim2();
                APtemp[2, 1] = APlim2[2];
                APtemp[3, 1] = APlim2[3];
                if (this.sentaterrizaje == -1)
                {
```

```
                    APtemp[2, 0] = APlim2[1];
                    APtemp[3, 0] = APlim2[1];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getpseclong()
* dp.Getpsecpen() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[0, 1] + Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        a = i;
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[1, 1] - Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        tt = t;
                        i = i + 1;
                        t = t + 1;
                    }
                    for (double tempz = this.tempzs[t - 1]; tempz <=
(dp.Getpseclong() * dp.Getpsecpen() / 100) + (dp.Getsseclong() * dp.Getssecpen()
/ 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[a, 0] - (this.tempzs[t] -
this.tempzs[tt]) * (100 / dp.Getssecpen());
                        APtemp[i, 1] = APtemp[a, 1] + Math.Abs(APtemp[i, 0] -
APtemp[a, 0]) * (dp.Getdivap() / 100);
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1 + a, 0] - (this.tempzs[t] -
this.tempzs[tt]) * (100 / dp.Getssecpen());
                        APtemp[i, 1] = APtemp[1 + a, 1] - Math.Abs(APtemp[i, 0] -
APtemp[a, 0]) * (dp.Getdivap() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
                else
                {
                    APtemp[2, 0] = APlim2[0];
                    APtemp[3, 0] = APlim2[0];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getpseclong()
* dp.Getpsecpen() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[0, 1] + Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        a = i;
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[1, 1] - Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
```

```
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        tt = t;
                        t = t + 1;
                    }
                    for (double tempz = this.tempzs[t-1]; tempz <=
(dp.Getpseclong() * dp.Getpsecpen() / 100) + (dp.Getsseclong() * dp.Getssecpen()
/ 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[0 + a, 0] + (this.tempzs[t] -
this.tempzs[tt]) * (100 / dp.Getssecpen());
                        APtemp[i, 1] = APtemp[0 + a, 1] + Math.Abs(APtemp[i, 0] -
APtemp[a, 0]) * (dp.Getdivap() / 100);
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1 + a, 0] + (this.tempzs[t] -
this.tempzs[tt]) * (100 / dp.Getssecpen());
                        APtemp[i, 1] = APtemp[1 + a, 1] - Math.Abs(APtemp[i, 0] -
APtemp[a, 0]) * (dp.Getdivap() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
                this.tempzs[t] = (dp.Getpseclong() * dp.Getpsecpen() / 100) +
(dp.Getsseclong() * dp.Getssecpen() / 100) + temppistapos[2];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = APtemp[3, 0];
                zstemp[t, 2] = temppistapos[1];
            }
            else
            {
                APtemp[2, 1] = APlim[2];
                APtemp[3, 1] = APlim[3];
                if (this.sentaterrizaje == -1)
                {
                    APtemp[2, 0] = APlim[1];
                    APtemp[3, 0] = APlim[1];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getpseclong()
* dp.Getpsecpen() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[0, 1] + Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[1, 1] - Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
```

```
                else
                {
                    APtemp[2, 0] = APlim[0];
                    APtemp[3, 0] = APlim[0];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getpseclong()
* dp.Getpsecpen() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        APtemp[i, 0] = APtemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[0, 1] + Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        i = i + 1;
                        APtemp[i, 0] = APtemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpsecpen());
                        APtemp[i, 1] = APtemp[1, 1] - Math.Abs(APtemp[i, 0] -
APtemp[0, 0]) * (dp.Getdivap() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = APtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
                this.tempzs[t] = (dp.Getpseclong() * dp.Getpsecpen() / 100) +
temppistapos[2];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = APtemp[3, 0];
                zstemp[t, 2] = temppistapos[1];
            }
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = APtemp[0, 0];
            zstemp[0, 2] = temppistapos[1];
            //Matriz que contendrá los puntos base de la SUPERFICIE DE
APROXIMACIÓN
            double[,] AP = new double[i, 2];
            //Trasladamos a la orientación original, en caso de que se haya
cargado un mapa de fondo se tendrá en cuenta la escala.
            for (int k = 0; k < i; k++)
            {
                double[] girAP = new double[2];
                girAP = its.giroinv(APtemp[k, 0], APtemp[k, 1], ang);
                AP[k, 0] = (girAP[0]) * this.escala[0] / this.escala[1];
                AP[k, 1] = (girAP[1]) * this.escala[0] / this.escala[1];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t + 1, 3];
            for (int j = 0; j < t + 1; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
                this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return AP;
        }

        //TRANSICIÓN
        public double [,] GetTR()
        {
```

```
            //MATRIZ que contendrá los puntos claves de la superfície de
transición
            double[,] TRtemp = new double[100, 2];
            bool[] v = its.inttrans(dp, fjd, temppistapos, this.pos,
this.sentaterrizaje, this.longitudpista, this.aprox, this.posumbral);
            //Guardamos los límites en un valor temporal.
            double[] TRlim = new double[4];
            TRlim = its.Getlim();
            TRtemp[0, 1] = TRlim[2];
            TRtemp[2, 1] = TRlim[2];
            TRtemp[1, 1] = this.temppistapos[1] + this.fjd.Getfrananch();
            TRtemp[3, 1] = this.temppistapos[1] + this.fjd.Getfrananch();
            TRtemp[4, 1] = this.temppistapos[1] - this.fjd.Getfrananch();
            TRtemp[7, 1] = this.temppistapos[1] - this.fjd.Getfrananch();
            TRtemp[5, 1] = TRlim[3];
            TRtemp[6, 1] = TRlim[3];
            //Definimos 2 matrices temoorales 1 para la parte de arriba y otra
para la parte de abajo.
            double[,] TrItar = new double[100, 2];
            double[,] TrItab = new double[100, 2];
            //Variable que indicará los puntos definidos para poder representar
las curvas de nivel
            int i = 0;
            //Variable que indicará las curvas de nivel que se realizarán
            int t = 1;
            //Matrices temporal que contendrá las coordenadas para los
letreros(tendrán que ser girados)
            double[,] templet = new double[100, 2];
            double[,] templett = new double[100, 2];
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(temppistapos[2]);
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            if (this.sentaterrizaje == -1)
            {
                TRtemp[2, 0] = TRlim[1];
                TRtemp[6, 0] = TRlim[1];
                TRtemp[0, 0] = TRlim[0];
                TRtemp[1, 0] = TRlim[0];
                TRtemp[4, 0] = TRlim[0];
                TRtemp[5, 0] = TRlim[0];
                TRtemp[3, 0] = (this.temppistapos[0] - longitudpista / 2 -
dp.Getdistuap() + posumbral);
                TRtemp[7, 0] = (this.temppistapos[0] - longitudpista / 2 -
dp.Getdistuap() + posumbral);
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
```

```
                    this.tempzs[t] = tempz;
                    TrItar[i, 1] = temppistapos[1] + fjd.Getfrananch() +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItab[i, 1] = (temppistapos[1] - fjd.Getfrananch()) -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItar[i, 0] = TRtemp[0, 0];
                    TrItab[i, 0] = TRtemp[0, 0];
                    templet[t, 0] = temppistapos[0];
                    templet[t, 1] = TrItar[i, 1];
                    templett[t, 0] = temppistapos[0];
                    templett[t, 1] = TrItab[i, 1];
                    i = i + 1;
                    TrItar[i, 1] = temppistapos[1] + fjd.Getfrananch() +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItar[i, 0] = TRtemp[3, 0] - Math.Abs(TrItar[i, 1] -
TRtemp[1, 1]) * 100 / dp.Getdivap();
                    TrItab[i, 1] = temppistapos[1] - fjd.Getfrananch() -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItab[i, 0] = TRtemp[7, 0] - Math.Abs(TrItab[i, 1] -
TRtemp[4, 1]) * 100 / dp.Getdivap();
                    i = i + 1;
                    t = t + 1;
                }
            }
            else
            {
                TRtemp[2, 0] = TRlim[0];
                TRtemp[6, 0] = TRlim[0];
                TRtemp[0, 0] = TRlim[1];
                TRtemp[1, 0] = TRlim[1];
                TRtemp[4, 0] = TRlim[1];
                TRtemp[5, 0] = TRlim[1];
                TRtemp[3, 0] = (this.temppistapos[0] + longitudpista / 2 +
dp.Getdistuap() - posumbral);
                TRtemp[7, 0] = (this.temppistapos[0] + longitudpista / 2 +
dp.Getdistuap() - posumbral);
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    TrItar[i, 1] = temppistapos[1] + fjd.Getfrananch() +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItab[i, 1] = (temppistapos[1] - fjd.Getfrananch()) -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItar[i, 0] = TRtemp[0, 0];
                    TrItab[i, 0] = TRtemp[0, 0];
                    templet[t, 0] = temppistapos[0];
                    templet[t, 1] = TrItar[i, 1];
                    templett[t, 0] = temppistapos[0];
                    templett[t, 1] = TrItab[i, 1];
                    i = i + 1;
                    TrItar[i, 1] = temppistapos[1] + fjd.Getfrananch() +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItar[i, 0] = TRtemp[3, 0] + Math.Abs(TrItar[i, 1] -
TRtemp[1, 1]) * 100 / dp.Getdivap();
                    TrItab[i, 1] = temppistapos[1] - fjd.Getfrananch() -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentrans();
                    TrItab[i, 0] = TRtemp[7, 0] + Math.Abs(TrItab[i, 1] -
TRtemp[4, 1]) * 100 / dp.Getdivap();
                    i = i + 1;
                    t = t + 1;
                }
```

```
            }
            this.tempzs[t] = 45 + temppistapos[2];
            templet[0, 0] = temppistapos[0];
            templet[0, 1] = TRtemp[1, 1];
            templett[0, 0] = temppistapos[0];
            templett[0, 1] = TRtemp[4, 1];
            int j = 8;
            for (int k = 0; k < i; k++)
            {
                TRtemp[j, 0] = TrItar[k, 0];
                TRtemp[j, 1] = TrItar[k, 1];
                j = j + 1;
            }
            for (int k = 0; k < i; k++)
            {
                TRtemp[j, 0] = TrItab[k, 0];
                TRtemp[j, 1] = TrItab[k, 1];
                j = j + 1;
            }
            //MATRIZ que contendrá los puntos claves de la superfície de
transición
            double[,] TR = new double[j, 2];
            for (int k = 0; k < j; k++)
            {
                double[] girTR = new double[2];
                girTR = its.giroinv(TRtemp[k, 0], TRtemp[k, 1], ang);
                TR[k, 0] = (girTR[0]) * this.escala[0] / this.escala[1];
                TR[k, 1] = (girTR[1]) * this.escala[0] / this.escala[1];
            }
            //Definimos la longitud de la matriz que contendrá las alturas de las
curvas de nivel y lo rellenamos con las alturas del vector temporal
            this.zs = new double[t * 2 + 2, 3];
            for (int k = 0; k < t * 2 + 2; k++)
            {
                if (k <= t)
                {
                    this.zs[k, 0] = this.tempzs[k];
                }
                else
                {
                    this.zs[k, 0] = this.tempzs[k - t - 1];
                }
                if (k == 0 || k == t + 1)
                {
                    if (k == 0)
                    {
                        this.zs[0, 1] = its.giroinv(templet[0, 0], templet[0, 1],
ang)[0];
                        this.zs[0, 2] = its.giroinv(templet[0, 0], templet[0, 1],
ang)[1];
                    }
                    else
                    {
                        this.zs[t + 1, 1] = its.giroinv(templett[0, 0],
templett[0, 1], ang)[0];
                        this.zs[t + 1, 2] = its.giroinv(templett[0, 0],
templett[0, 1], ang)[1];
                    }
                }
                else
                {
                    double[] girtemplet = new double[2];
```

```
                    if (k <= t)
                    {
                        if (k != t)
                        {
                            girtemplet = its.giroinv(templet[k, 0], templet[k,
1], ang);
                            this.zs[k, 1] = girtemplet[0] * this.escala[0] /
this.escala[1];
                            this.zs[k, 2] = girtemplet[1] * this.escala[0] /
this.escala[1];
                        }
                        else
                        {
                            this.zs[k, 1] = its.giroinv(temppistapos[0],
TRtemp[0, 1], ang)[0];
                            this.zs[k, 2] = its.giroinv(temppistapos[0],
TRtemp[0, 1], ang)[1];
                        }
                    }
                    else
                    {
                        if (k != t * 2 + 1)
                        {
                            girtemplet = its.giroinv(templett[k - t - 1, 0],
templett[k - t - 1, 1], ang);
                            this.zs[k, 1] = girtemplet[0] * this.escala[0] /
this.escala[1];
                            this.zs[k, 2] = girtemplet[1] * this.escala[0] /
this.escala[1];
                        }
                        else
                        {
                            this.zs[k, 1] = its.giroinv(temppistapos[0],
TRtemp[5, 1], ang)[0];
                            this.zs[k, 2] = its.giroinv(temppistapos[0],
TRtemp[5, 1], ang)[1];
                        }
                    }
                }
            }
            return TR;
        }

        //SUPERFÍCIE DE ATERRIZAJE INTERRUMPIDO
        public double[,] GetSAI()
        {
            //Matrices temporales para los puntos del contorno y los letreros de
las profundidades
            double[,] SAItemp = new double[100, 2];
            double[,] zstemp = new double[100, 3];
            bool v = its.intsai(dp, this.pos, temppistapos, this.longitudpista,
this.sentaterrizaje, this.posumbral, this.aprox);
            double[] SAIlim = new double[4];
            SAIlim = its.Getlim();
            SAItemp[0, 1] = this.temppistapos[1] + dp.Getlongbisai() / 2;
            SAItemp[2, 1] = SAIlim[2];
            SAItemp[1, 1] = this.temppistapos[1] - dp.Getlongbisai() / 2;
            SAItemp[3, 1] = SAIlim[3];
            int i = 4;
            int t = 1;
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
```

```
            this.tempzs[0] = Math.Round(temppistapos[2]);
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            if (this.sentaterrizaje == -1)
            {
                SAItemp[0, 0] = SAIlim[1];
                SAItemp[1, 0] = SAIlim[1];
                SAItemp[2, 0] = SAIlim[0];
                SAItemp[3, 0] = SAIlim[0];
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    SAItemp[i, 0] = SAItemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpensai());
                    SAItemp[i, 1] = SAItemp[0, 1] + Math.Abs(SAItemp[i, 0] -
SAItemp[0, 0]) * (dp.Getdivsai() / 100);
                    i = i + 1;
                    SAItemp[i, 0] = SAItemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpensai());
                    SAItemp[i, 1] = SAItemp[1, 1] - Math.Abs(SAItemp[i, 0] -
SAItemp[0, 0]) * (dp.Getdivsai() / 100);
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = SAItemp[i, 0];
                    zstemp[t, 2] = temppistapos[1];
                    i = i + 1;
                    t = t + 1;
                }
            }
            else
            {
                SAItemp[0, 0] = SAIlim[0];
                SAItemp[1, 0] = SAIlim[0];
                SAItemp[2, 0] = SAIlim[1];
                SAItemp[3, 0] = SAIlim[1];
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    SAItemp[i, 0] = SAItemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpensai());
                    SAItemp[i, 1] = SAItemp[0, 1] + Math.Abs(SAItemp[i, 0] -
SAItemp[0, 0]) * (dp.Getdivsai() / 100);
                    i = i + 1;
                    SAItemp[i, 0] = SAItemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpensai());
                    SAItemp[i, 1] = SAItemp[1, 1] - Math.Abs(SAItemp[i, 0] -
SAItemp[0, 0]) * (dp.Getdivsai() / 100);
                    zstemp[t, 0] = this.tempzs[t];
```

```
                    zstemp[t, 1] = SAItemp[i, 0];
                    zstemp[t, 2] = temppistapos[1];
                    i = i + 1;
                    t = t + 1;
                }
            }
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = SAItemp[0, 0];
            zstemp[0, 2] = temppistapos[1];
            this.tempzs[t] = 45 + temppistapos[2];
            zstemp[t, 0] = this.tempzs[t];
            zstemp[t, 1] = SAItemp[2, 0];
            zstemp[t, 2] = temppistapos[1];
            //MATRIZ que contendrá los puntos clave de la SAI
            double[,] SAI = new double[i, 2];
            for (int j = 0; j < i; j++)
            {
                double[] girSAI = new double[2];
                girSAI = its.giroinv(SAItemp[j, 0], SAItemp[j, 1], ang);
                SAI[j, 0] = (girSAI[0]) * this.escala[0] / this.escala[1];
                SAI[j, 1] = (girSAI[1]) * this.escala[0] / this.escala[1];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t + 1, 3];
            for (int j = 0; j < t + 1; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
                this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return SAI;
        }

        //SUPERFICIE DE ASCENSO EN EL DESPEGUE
        public double[,] GetASCD()
        {
            //Matrices temporales para los puntos del contorno y los letreros de
las profundidades
            double [,] ASCDtemp=new double [150,2];
            double[,] zstemp = new double[150, 3];
            bool v = its.intasc(dp, this.pos, temppistapos, this.longitudpista,
this.sentdespegue);
            double[] ASCDlim = new double[4];
            ASCDlim = its.Getlim();
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(temppistapos[2]);
            int i = 0;
            int t = 1;
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
```

```
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            ASCDtemp[0, 1] = this.temppistapos[1] + dp.Getlongbiascd() / 2;
            ASCDtemp[1, 1] = this.temppistapos[1] - dp.Getlongbiascd() / 2;
            if (longitudpista >= 1200)
            {
                double xx2 = its.Getxxs()[1];
                ASCDtemp[5, 1] = ASCDlim[2];
                ASCDtemp[4, 1] = ASCDlim[2];
                ASCDtemp[3, 1] = ASCDlim[3];
                ASCDtemp[2, 1] = ASCDlim[3];
                ASCDtemp[5, 0] = xx2;
                ASCDtemp[2, 0] = xx2;
                i = 6;
                if (this.sentdespegue == 1)
                {
                    ASCDtemp[0, 0] = ASCDlim[1];
                    ASCDtemp[1, 0] = ASCDlim[1];
                    ASCDtemp[4, 0] = ASCDlim[0];
                    ASCDtemp[3, 0] = ASCDlim[0];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getlongascd()
* dp.Getpenascd() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        ASCDtemp[i, 0] = ASCDtemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                        if (ASCDtemp[i, 0] < ASCDtemp[5, 0])
                        {
                            ASCDtemp[i, 1] = ASCDtemp[0, 1] +
Math.Abs(ASCDtemp[i, 0] - ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                            i = i + 1;
                            ASCDtemp[i, 0] = ASCDtemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                            ASCDtemp[i, 1] = ASCDtemp[1, 1] -
Math.Abs(ASCDtemp[i, 0] - ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                        }
                        else
                        {
                            ASCDtemp[i, 1] = ASCDtemp[4, 1];
                            i = i + 1;
                            ASCDtemp[i, 0] = ASCDtemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                            ASCDtemp[i, 1] = ASCDtemp[3, 1];
                        }
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = ASCDtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
                else
                {
                    ASCDtemp[0, 0] = ASCDlim[0];
                    ASCDtemp[1, 0] = ASCDlim[0];
                    ASCDtemp[4, 0] = ASCDlim[1];
                    ASCDtemp[3, 0] = ASCDlim[1];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getlongascd()
* dp.Getpenascd() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
```

```
                        this.tempzs[t] = tempz;
                        ASCDtemp[i, 0] = ASCDtemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                        if (ASCDtemp[i, 0] > ASCDtemp[5, 0])
                        {
                            ASCDtemp[i, 1] = ASCDtemp[0, 1] +
Math.Abs(ASCDtemp[i, 0] - ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                            i = i + 1;
                            ASCDtemp[i, 0] = ASCDtemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                            ASCDtemp[i, 1] = ASCDtemp[1, 1] -
Math.Abs(ASCDtemp[i, 0] - ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                        }
                        else
                        {
                            ASCDtemp[i, 1] = ASCDtemp[4, 1];
                            i = i + 1;
                            ASCDtemp[i, 0] = ASCDtemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                            ASCDtemp[i, 1] = ASCDtemp[3, 1];
                        }
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = ASCDtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
                        t = t + 1;
                    }
                }
                this.tempzs[t] = (dp.Getlongascd() * dp.Getpenascd() / 100) +
temppistapos[2];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = ASCDtemp[4, 0];
                zstemp[t, 2] = temppistapos[1];
            }
            else
            {
                ASCDtemp[2, 1] = ASCDlim[2];
                ASCDtemp[3, 1] = ASCDlim[3];
                i = 4;
                if (this.sentdespegue == 1)
                {
                    ASCDtemp[0, 0] = ASCDlim[1];
                    ASCDtemp[1, 0] = ASCDlim[1];
                    ASCDtemp[2, 0] = ASCDlim[0];
                    ASCDtemp[3, 0] = ASCDlim[0];
                    for (double tempz = this.tempzs[1]; tempz < (dp.Getlongascd()
* dp.Getpenascd() / 100) + temppistapos[2]; tempz = tempz + 5)
                    {
                        this.tempzs[t] = tempz;
                        ASCDtemp[i, 0] = ASCDtemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                        ASCDtemp[i, 1] = ASCDtemp[0, 1] + Math.Abs(ASCDtemp[i, 0]
- ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                        i = i + 1;
                        ASCDtemp[i, 0] = ASCDtemp[1, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                        ASCDtemp[i, 1] = ASCDtemp[1, 1] - Math.Abs(ASCDtemp[i, 0]
- ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                        zstemp[t, 0] = this.tempzs[t];
                        zstemp[t, 1] = ASCDtemp[i, 0];
                        zstemp[t, 2] = temppistapos[1];
                        i = i + 1;
```

```
                    t = t + 1;
                }
                this.tempzs[t] = (dp.Getlongascd() * dp.Getpenascd() / 100) +
temppistapos[2];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = ASCDtemp[2, 0];
                zstemp[t, 2] = temppistapos[1];
            }
            else
            {
                ASCDtemp[0, 0] = ASCDlim[0];
                ASCDtemp[1, 0] = ASCDlim[0];
                ASCDtemp[2, 0] = ASCDlim[1];
                ASCDtemp[3, 0] = ASCDlim[1];
                for (double tempz = this.tempzs[1]; tempz < (dp.Getlongascd()
* dp.Getpenascd() / 100) + temppistapos[2]; tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    ASCDtemp[i, 0] = ASCDtemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                    ASCDtemp[i, 1] = ASCDtemp[0, 1] + Math.Abs(ASCDtemp[i, 0]
- ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                    i = i + 1;
                    ASCDtemp[i, 0] = ASCDtemp[1, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenascd());
                    ASCDtemp[i, 1] = ASCDtemp[1, 1] - Math.Abs(ASCDtemp[i, 0]
- ASCDtemp[0, 0]) * (dp.Getdivascd() / 100);
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = ASCDtemp[i, 0];
                    zstemp[t, 2] = temppistapos[1];
                    i = i + 1;
                    t = t + 1;
                }
                this.tempzs[t] = (dp.Getlongascd() * dp.Getpenascd() / 100) +
temppistapos[2];
                zstemp[t, 0] = this.tempzs[t];
                zstemp[t, 1] = ASCDtemp[2, 0];
                zstemp[t, 2] = temppistapos[1];
            }
        }
        zstemp[0, 0] = this.tempzs[0];
        zstemp[0, 1] = ASCDtemp[0, 0];
        zstemp[0, 2] = temppistapos[1];
        double[,] ASCD = new double[i, 2];
        for (int k = 0; k < i; k++)
        {
            double[] girASCD = new double[2];
            girASCD = its.giroinv(ASCDtemp[k, 0], ASCDtemp[k, 1], ang);
            ASCD[k, 0] = (girASCD[0]) * this.escala[0] / this.escala[1];
            ASCD[k, 1] = (girASCD[1]) * this.escala[0] / this.escala[1];
        }
        //Trasladamos las coordenadas de la matriz zs
        this.zs = new double[t + 1, 3];
        for (int j = 0; j < t + 1; j++)
        {
            this.zs[j, 0] = zstemp[j, 0];
            double[] girzs = new double[2];
            girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
            this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
            this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
        }
        return ASCD;
```

```
        }

        //APROXIMACIÓN INTERNA
        public double[,] GetApI()
        {
            //Matriz temporal que contendrá los puntos base de la SUPERFICIE DE
APROXIMACIÓN
            double[,] ApItemp = new double[100, 2];
            bool[] v = its.intaprox(this.aprox, dp, this.pos, temppistapos,
this.clave, this.longitudpista, this.posumbral, this.sentaterrizaje);
            //Guardamos los límites en un valor temporal.
            double[] ApIlim3 = new double[4];
            ApIlim3 = its.Getlim3();
            ApItemp[0, 1] = ApIlim3[2];
            ApItemp[2, 1] = ApIlim3[2];
            ApItemp[1, 1] = ApIlim3[3];
            ApItemp[3, 1] = ApIlim3[3];
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(temppistapos[2]);
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            double[,] zstemp = new double[100, 3];
            //Variable que indicará las curvas de nivel que se realizarán
            int t = 1;
            //Variable que indicará los puntos definidos para poder representar
las curvas de nivel
            int i = 4;
            //
            if (this.sentaterrizaje == -1)
            {
                ApItemp[2, 0] = ApIlim3[1];
                ApItemp[3, 0] = ApIlim3[1];
                ApItemp[0, 0] = ApIlim3[0];
                ApItemp[1, 0] = ApIlim3[0];
                for (double tempz = this.tempzs[1]; tempz < (dp.Getlongaint() *
dp.Getpenaint() / 100) + temppistapos[2]; tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    ApItemp[i, 1] = ApItemp[0, 1];
                    ApItemp[i, 0] = ApItemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenaint());
                    i = i + 1;
                    ApItemp[i, 1] = ApItemp[1, 1];
                    ApItemp[i, 0] = ApItemp[0, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenaint());
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = ApItemp[i, 0];
                    zstemp[t, 2] = temppistapos[1];
```

```
                    i = i + 1;
                    t = t + 1;
                }
            }
            else
            {
                ApItemp[2, 0] = ApIlim3[0];
                ApItemp[3, 0] = ApIlim3[0];
                ApItemp[0, 0] = ApIlim3[1];
                ApItemp[1, 0] = ApIlim3[1];
                for (double tempz = this.tempzs[1]; tempz < (dp.Getlongaint() *
dp.Getpenaint() / 100) + temppistapos[2]; tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    ApItemp[i, 1] = ApItemp[0, 1];
                    ApItemp[i, 0] = ApItemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenaint());
                    i = i + 1;
                    ApItemp[i, 1] = ApItemp[1, 1];
                    ApItemp[i, 0] = ApItemp[0, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / dp.Getpenaint());
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = ApItemp[i, 0];
                    zstemp[t, 2] = temppistapos[1];
                    i = i + 1;
                    t = t + 1;
                }
            }
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = ApItemp[0, 0];
            zstemp[0, 2] = temppistapos[1];
            this.tempzs[t] = (dp.Getlongaint() * dp.Getpenaint() / 100) +
temppistapos[2];
            zstemp[t, 0] = this.tempzs[t];
            zstemp[t, 1] = ApItemp[2, 0];
            zstemp[t, 2] = temppistapos[1];
            //Matriz que contendrá los puntos base de la SUPERFICIE DE
APROXIMACIÓN
            double[,] ApI = new double[i, 2];
            for (int j = 0; j < i; j++)
            {
                double[] girApI = new double[2];
                girApI = its.giroinv(ApItemp[j, 0], ApItemp[j, 1], ang);
                ApI[j, 0] = (girApI[0]) * this.escala[0] / this.escala[1];
                ApI[j, 1] = (girApI[1]) * this.escala[0] / this.escala[1];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t + 1, 3];
            for (int j = 0; j < t+1; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
                this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return ApI;
        }


        //TRANSICIÓN INTERNA
        public double[,] GetTrI()
```

```
        {
            //Matriz que contendrá los puntos base de la SUPERFICIE DE TRANSICIÓN
INTERNA
            double[,] TrItemp = new double[100, 2];
            bool[] v = its.inttrans(dp, fjd, temppistapos, this.pos,
this.sentaterrizaje, this.longitudpista, this.aprox, this.posumbral);
            //Guardamos los límites en un valor temporal.
            double[] TrIlim3 = new double[4];
            TrIlim3 = its.Getlim3();
            TrItemp[3, 1] = TrIlim3[2];
            TrItemp[2, 1] = TrIlim3[2];
            TrItemp[0, 1] = temppistapos[1] + dp.Getanchapint()/2;
            TrItemp[4, 1] = temppistapos[1] + dp.Getanchapint()/2;
            TrItemp[5, 1] = temppistapos[1] - dp.Getanchapint()/2;
            TrItemp[9, 1] = temppistapos[1] - dp.Getanchapint()/2;
            TrItemp[6, 1] = TrIlim3[3];
            TrItemp[7, 1] = TrIlim3[3];
            TrItemp[2, 0] = its.Getxxs()[0];
            TrItemp[7, 0] = its.Getxxs()[0];
            TrItemp[4, 0] = its.Getxxs()[1];
            TrItemp[5, 0] = its.Getxxs()[1];
            //Definimos 2 matrices temoorales 1 para la parte de arriba y otra
para la parte de abajo.
            double[,] TrItar = new double[100, 2];
            double[,] TrItab = new double[100, 2];
            //Variable que indicará los puntos definidos para poder representar
las curvas de nivel
            int i = 0;
            //Variable que indicará las curvas de nivel que se realizarán
            int t = 1;
            //Matrices temporal que contendrá las coordenadas para los
letreros(tendrán que ser girados)
            double[,] templet = new double[100, 2];
            double[,] templett = new double[100, 2];
            if (this.sentaterrizaje == -1)
            {
                TrItemp[1, 1] = TrIlim3[2] + Math.Abs(TrIlim3[1] -
its.Getxxs()[0]) * (-dp.Getdivsai() / 100);
                TrItemp[8, 1] = TrIlim3[3] + Math.Abs(TrIlim3[1] -
its.Getxxs()[0]) * (dp.Getdivsai() / 100);
                TrItemp[0, 0] = TrIlim3[1];
                TrItemp[1, 0] = TrIlim3[1];
                TrItemp[8, 0] = TrIlim3[1];
                TrItemp[9, 0] = TrIlim3[1];
                TrItemp[3, 0] = TrIlim3[0];
                TrItemp[6, 0] = TrIlim3[0];
                string z = Convert.ToString(Math.Round(temppistapos[2], 0));
                double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
                this.tempzs[0] = Math.Round(temppistapos[2]);
                if (Math.Round(temppistapos[2]) % 5 == 0)
                {
                    this.tempzs[1] = this.tempzs[0] + 5;
                }
                else
                {
                    if (zz < 5)
                    {
                        this.tempzs[1] = this.tempzs[0] + (5 - zz);
                    }
                    else
                    {
                        this.tempzs[1] = this.tempzs[0] + (10 - zz);
```

```
                    }
                }
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    TrItar[i, 1] = temppistapos[1] + dp.Getanchapint()/2 +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItab[i, 1] = (temppistapos[1] - dp.Getanchapint()/2) -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    if (TrItar[i, 1] <= TrItemp[1, 1])
                    {
                        TrItar[i, 0] = TrIlim3[1];
                        TrItab[i, 0] = TrIlim3[1];
                    }
                    else
                    {
                        TrItar[i, 0] = TrItemp[1, 0] + (TrItar[i, 1] - TrItemp[1,
1]) * (100 / dp.Getdivsai());
                        TrItab[i, 0] = TrItemp[8, 0] + (-TrItab[i, 1] +
TrItemp[8, 1]) * (100 / dp.Getdivsai());
                    }
                    templet[t, 0] = TrItemp[2, 0];
                    templet[t, 1] = TrItar[i, 1];
                    templett[t, 0] = TrItemp[2, 0];
                    templett[t, 1] = TrItab[i, 1];
                    i = i + 1;
                    TrItar[i, 1] = temppistapos[1] + dp.Getanchapint()/2 +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItar[i, 0] = its.Getxxs()[1] + (-TrItemp[4, 1] + TrItar[i,
1]) * 100 / dp.Getdivsai();
                    TrItab[i, 1] = temppistapos[1] - dp.Getanchapint()/2 -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItab[i, 0] = its.Getxxs()[1] + (TrItemp[5, 1] - TrItab[i,
1]) * 100 / dp.Getdivsai();
                    i = i + 1;
                    t = t + 1;
                }
                this.tempzs[t] = 45 + temppistapos[2];
            }
            else
            {
                TrItemp[1, 1] = TrIlim3[2] + Math.Abs(TrIlim3[0] -
its.Getxxs()[0]) * (-dp.Getdivsai() / 100);
                TrItemp[8, 1] = TrIlim3[3] + Math.Abs(TrIlim3[0] -
its.Getxxs()[0]) * (dp.Getdivsai() / 100);
                TrItemp[0, 0] = TrIlim3[0];
                TrItemp[1, 0] = TrIlim3[0];
                TrItemp[8, 0] = TrIlim3[0];
                TrItemp[9, 0] = TrIlim3[0];
                TrItemp[3, 0] = TrIlim3[1];
                TrItemp[6, 0] = TrIlim3[1];
                string z = Convert.ToString(Math.Round(temppistapos[2], 0));
                double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
                this.tempzs[0] = Math.Round(temppistapos[2]);
                if (Math.Round(temppistapos[2]) % 5 == 0)
                {
                    this.tempzs[1] = this.tempzs[0] + 5;
                }
                else
                {
                    if (zz < 5)
```

```
                    {
                        this.tempzs[1] = this.tempzs[0] + (5 - zz);
                    }
                    else
                    {
                        this.tempzs[1] = this.tempzs[0] + (10 - zz);
                    }
                }
                for (double tempz = this.tempzs[1]; tempz < 45 + temppistapos[2];
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    TrItar[i, 1] = temppistapos[1] + dp.Getanchapint()/2 +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItab[i, 1] = (temppistapos[1] - dp.Getanchapint()/2) -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    if (TrItar[i, 1] <= TrItemp[1, 1])
                    {
                        TrItar[i, 0] = TrIlim3[0];
                        TrItab[i, 0] = TrIlim3[0];
                    }
                    else
                    {
                        TrItar[i, 0] = TrItemp[1, 0] - (TrItar[i, 1] - TrItemp[1,
1]) * (100 / dp.Getdivsai());
                        TrItab[i, 0] = TrItemp[8, 0] - (-TrItab[i, 1] +
TrItemp[8, 1]) * (100 / dp.Getdivsai());
                    }
                    templet[t, 0] = TrItemp[2, 0];
                    templet[t, 1] = TrItar[i, 1];
                    templett[t, 0] = TrItemp[2, 0];
                    templett[t, 1] = TrItab[i, 1];
                    i = i + 1;
                    TrItar[i, 1] = temppistapos[1] + dp.Getanchapint()/2 +
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItar[i, 0] = its.Getxxs()[1] - (-TrItemp[4, 1] + TrItar[i,
1]) * 100 / dp.Getdivsai();
                    TrItab[i, 1] = temppistapos[1] - dp.Getanchapint()/2 -
(this.tempzs[t] - this.tempzs[0]) * 100 / dp.Getpentransint();
                    TrItab[i, 0] = its.Getxxs()[1] - (TrItemp[5, 1] - TrItab[i,
1]) * 100 / dp.Getdivsai();
                    i = i + 1;
                    t = t + 1;
                }
                this.tempzs[t] = 45 + temppistapos[2];
            }
            templet[0, 0] = TrItemp[2, 0];
            templet[0, 1] = TrItemp[0, 1];
            templett[0, 0] = TrItemp[2, 0];
            templett[0, 1] = TrItemp[9, 1];
            int j=10;
            for (int k = 0; k < i; k++)
            {
                TrItemp[j, 0] = TrItar[k, 0];
                TrItemp[j, 1] = TrItar[k, 1];
                j = j + 1;
            }
            for (int k = 0; k < i; k++)
            {
                TrItemp[j, 0] = TrItab[k, 0];
                TrItemp[j, 1] = TrItab[k, 1];
                j = j + 1;
```

```
            }
            //Ahora lo pasamos todo a la matriz principal que contendrá todos los
puntos con la orientación original.
            double[,] TrI = new double[j, 2];
            for (int k = 0; k < j; k++)
            {
                double[] girTrI = new double[2];
                girTrI = its.giroinv(TrItemp[k, 0], TrItemp[k, 1], ang);
                TrI[k, 0] = (girTrI[0]) * this.escala[0] / this.escala[1];
                TrI[k, 1] = (girTrI[1]) * this.escala[0] / this.escala[1];
            }
            //Definimos la longitud de la matriz que contendrá las alturas de las
curvas de nivel y lo rellenamos con las alturas del vector temporal
            this.zs = new double[t * 2 + 2, 3];
            for (int k = 0; k < t * 2 + 2; k++)
            {
                if (k <= t)
                {
                    this.zs[k, 0] = this.tempzs[k];
                }
                else
                {
                    this.zs[k, 0] = this.tempzs[k - t - 1];
                }
                if (k == 0 || k == t+1)
                {
                    if (k == 0)
                    {
                        this.zs[0, 1] = its.giroinv(templet[0, 0], templet[0, 1],
ang)[0];
                        this.zs[0, 2] = its.giroinv(templet[0, 0], templet[0, 1],
ang)[1];
                    }
                    else
                    {
                        this.zs[t + 1, 1] = its.giroinv(templett[0, 0],
templett[0, 1], ang)[0];
                        this.zs[t + 1, 2] = its.giroinv(templett[0, 0],
templett[0, 1], ang)[1];
                    }
                }
                else
                {
                    double[] girtemplet = new double[2];
                    if (k <= t)
                    {
                        if (k != t)
                        {
                            girtemplet = its.giroinv(templet[k, 0], templet[k,
1], ang);
                            this.zs[k, 1] = girtemplet[0] * this.escala[0] /
this.escala[1];
                            this.zs[k, 2] = girtemplet[1] * this.escala[0] /
this.escala[1];
                        }
                        else
                        {
                            this.zs[k, 1] = TrI[2, 0];
                            this.zs[k, 2] = TrI[2, 1];
                        }
                    }
                    else
```

```
                    {
                        if (k != t * 2 + 1)
                        {
                            girtemplet = its.giroinv(templett[k - t - 1, 0],
templett[k - t - 1, 1], ang);
                            this.zs[k, 1] = girtemplet[0] * this.escala[0] /
this.escala[1];
                            this.zs[k, 2] = girtemplet[1] * this.escala[0] /
this.escala[1];
                        }
                        else
                        {
                            this.zs[k, 1] = TrI[7, 0];
                            this.zs[k, 2] = TrI[7, 1];
                        }
                    }
                }
            }
            return TrI;
        }
        public void Setsrad(SRad srad)
        {
            this.srad = srad;
            this.radpos = new double[3];
            this.radpos[0] = this.srad.Getradpos()[0];
            this.radpos[1] = this.srad.Getradpos()[1];
            this.radpos[2] = this.srad.Getradpos()[2];
            //Trasladamos las coordenadas para tener unas superfícies paralelas
al eje horizontal.
            this.radpos = its.giro(this.radpos, this.ang);
        }

        //ILS/GP
        public double[,] GetIGp()
        {
            //Matriz que contendrá los puntos base de la SERVIDUMBRE DEL GP
            double[,] IGptemp = new double[100, 2];
            double[,] zstemp = new double[100, 3];
            bool v = its.igpint(this.pos, this.radpos, this.posumbral,
this.longitudpista, temppistapos, this.ancho, this.ang);
            //Guardamos los límites en un valor temporal.
            double[] Igplim = new double[4];
            Igplim = its.Getlim();
            double[] Igpxxs = new double[4];
            Igpxxs = its.Getxxs();
            double[] Igpxyils = new double[4];
            Igpxyils = its.Getxyils();
            double[] ya = its.Getya();
            IGptemp[7, 1] = Igplim[2];
            IGptemp[6, 1] = Igplim[3];
            IGptemp[1, 1] = Igpxyils[2];
            IGptemp[2, 1] = Igpxyils[2];
            IGptemp[3, 1] = Igpxyils[3];
            IGptemp[0, 1] = Igpxyils[3];
            IGptemp[4, 0] = Igpxxs[0];
            IGptemp[4, 1] = Igpxxs[2];
            IGptemp[5, 0] = Igpxxs[1];
            IGptemp[5, 1] = Igpxxs[3];
            int i = 8;
            int t = 1;
            string z = Convert.ToString(Math.Round(temppistapos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
```

```
            this.tempzs[0] = Math.Round(temppistapos[2]);
            if (Math.Round(temppistapos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            if (this.radpos[0] < temppistapos[0])
            {
                IGptemp[7, 0] = Igplim[1];
                IGptemp[6, 0] = Igplim[1];
                IGptemp[2, 0] = Igpxyils[0];
                IGptemp[3, 0] = Igpxyils[0];
                IGptemp[0, 0] = Igpxyils[1];
                IGptemp[1, 0] = Igpxyils[1];
                for (double tempz = this.tempzs[1]; tempz < radpos[2] +
Math.Abs(IGptemp[7, 0] - IGptemp[2, 0]) * 2 / 100; tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    IGptemp[i, 0] = IGptemp[2, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / 2);
                    IGptemp[i, 1] = ya[2] * IGptemp[i, 0] + ya[4];
                    i = i + 1;
                    IGptemp[i, 0] = IGptemp[2, 0] - (this.tempzs[t] -
this.tempzs[0]) * (100 / 2);
                    IGptemp[i, 1] = ya[3] * IGptemp[i, 0] + ya[5];
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = IGptemp[i, 0];
                    zstemp[t, 2] = radpos[1];
                    i = i + 1;
                    t = t + 1;
                }
            }
            else
            {
                IGptemp[7, 0] = Igplim[0];
                IGptemp[6, 0] = Igplim[0];
                IGptemp[2, 0] = Igpxyils[1];
                IGptemp[3, 0] = Igpxyils[1];
                IGptemp[0, 0] = Igpxyils[0];
                IGptemp[1, 0] = Igpxyils[0];
                for (double tempz = this.tempzs[1]; tempz < radpos[2] +
Math.Abs(IGptemp[7, 0] - IGptemp[2, 0]) * 2 / 100; tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    IGptemp[i, 0] = IGptemp[2, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / 2);
                    IGptemp[i, 1] = ya[2] * IGptemp[i, 0] + ya[4];
                    i = i + 1;
                    IGptemp[i, 0] = IGptemp[2, 0] + (this.tempzs[t] -
this.tempzs[0]) * (100 / 2);
                    IGptemp[i, 1] = ya[3] * IGptemp[i, 0] + ya[5];
                    zstemp[t, 0] = this.tempzs[t];
```

```
                    zstemp[t, 1] = IGptemp[i, 0];
                    zstemp[t, 2] = radpos[1];
                    i = i + 1;
                    t = t + 1;
                }
            }
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = IGptemp[2, 0];
            zstemp[0, 2] = radpos[1];
            this.tempzs[t] = radpos[2] + Math.Abs(IGptemp[7, 0] - IGptemp[2, 0])
* 2 / 100;
            zstemp[t, 0] = Math.Round(this.tempzs[t],2);
            zstemp[t, 1] = IGptemp[7, 0];
            zstemp[t, 2] = radpos[1];
            double[,] IGp = new double[i, 2];
            for (int j = 0; j < i; j++)
            {
                double[] girIGp = new double[2];
                girIGp = its.giroinv(IGptemp[j, 0], IGptemp[j, 1], ang);
                IGp[j, 0] = (girIGp[0]) * this.escala[0] / this.escala[1];
                IGp[j, 1] = (girIGp[1]) * this.escala[0] / this.escala[1];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t + 1, 3];
            for (int j = 0; j < t + 1; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[j, 1], zstemp[j, 2], ang);
                this.zs[j, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[j, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return IGp;
        }

        //ILS/LOC
        public double[,] GetILoc()
        {
            //Matriz que contendrá los puntos base de la SERVIDUMBRE DEL
LOCALIZADOR
            double[,] ILoctemp = new double[200, 2];
            double[,] zstemp = new double[200, 3];
            bool v = its.ilocint(this.pos, this.srad.Getradpos(), this.posumbral,
this.sentaterrizaje, this.longitudpista, temppistapos, this.ang);
            //Guardamos los límites en un valor temporal.
            double[] Iloclim= new double[4];
            Iloclim = its.Getlim();
            //Guardamos los límites de la zona de seguridad
            double[] xyils = new double[4];
            xyils = its.Getxyils();
            double[] ya = its.Getya();
            ILoctemp[0, 1] = xyils[2];
            ILoctemp[1, 1] = xyils[2];
            ILoctemp[2, 1] = xyils[3];
            ILoctemp[3, 1] = xyils[3];
            ILoctemp[6, 1] = Iloclim[3];
            ILoctemp[4, 1] = Iloclim[3];
            ILoctemp[7, 1] = Iloclim[2];
            ILoctemp[5, 1] = Iloclim[2];
            ILoctemp[9, 1] = ya[1];
            ILoctemp[8, 1] = ya[1];
            ILoctemp[10, 1] = ya[0];
```

```
            ILoctemp[11, 1] = ya[0];
            ILoctemp[1, 0] = xyils[1];
            ILoctemp[2, 0] = xyils[1];
            ILoctemp[0, 0] = xyils[0];
            ILoctemp[3, 0] = xyils[0];
            ILoctemp[4, 0] = Iloclim[0];
            ILoctemp[5, 0] = Iloclim[0];
            ILoctemp[6, 0] = Iloclim[1];
            ILoctemp[7, 0] = Iloclim[1];
            ILoctemp[8, 0] = ya[6];
            ILoctemp[11, 0] = ya[6];
            ILoctemp[9, 0] = ya[7];
            ILoctemp[10, 0] = ya[7];
            //Curvas de nivel
            string z = Convert.ToString(Math.Round(radpos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(radpos[2]);
            if (Math.Round(radpos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            //Variable que indicará los puntos definidos para poder representar
las curvas de nivel
            int j = 0;
            int tt = 1;
            //Definimos 2 matrices temoorales 1 para la parte de derecha e
izquierda.
            double[,] tad = new double[100, 2];
            double[,] tai = new double[100, 2];
            //Matrices temporal que contendrá las coordenadas para los
letreros(tendrán que ser girados)
            double[,] templet1 = new double[100, 2];
            double[,] templett1 = new double[100, 2];
            for (double tempz = this.tempzs[1]; tempz < ((ILoctemp[5, 0] -
ILoctemp[3, 0]) * 2 / 100 + this.radpos[2]); tempz = tempz + 5)
            {
                this.tempzs[tt] = tempz;
                tad[j, 0] = ILoctemp[3, 0] + (this.tempzs[tt] - this.tempzs[0]) *
100 / 2;
                tai[j, 0] = ILoctemp[2, 0] - (this.tempzs[tt] - this.tempzs[0]) *
100 / 2;
                tad[j, 1] = ILoctemp[3, 1] + (tad[j, 0] - ILoctemp[3, 0]) *
(Math.Tan(30 * 2 * Math.PI / 360));
                tai[j, 1] = ILoctemp[2, 1] + Math.Abs(tai[j, 0] - ILoctemp[2, 0])
* (Math.Tan(30 * 2 * Math.PI / 360));
                templet1[tt, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
                templet1[tt, 0] = tad[j, 0];
                templett1[tt, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
                templett1[tt, 0] = tai[j, 0];
                j = j + 1;
```

```
                tad[j, 0] = ILoctemp[0, 0] + (this.tempzs[tt] - this.tempzs[0]) *
100 / 2;
                tai[j, 0] = ILoctemp[1, 0] - (this.tempzs[tt] - this.tempzs[0]) *
100 / 2;
                tad[j, 1] = ILoctemp[0, 1] - (tad[j, 0] - ILoctemp[0, 0]) *
(Math.Tan(30 * 2 * Math.PI / 360));
                tai[j, 1] = ILoctemp[1, 1] - Math.Abs(tai[j, 0] - ILoctemp[1, 0])
* (Math.Tan(30 * 2 * Math.PI / 360));
                j = j + 1;
                tt = tt + 1;
            }
            templet1[0, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
            templet1[0, 0] = ILoctemp[3, 0];
            templett1[0, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
            templett1[0, 0] = ILoctemp[1, 0];
            templet1[tt, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
            templet1[tt, 0] = ILoctemp[4, 0];
            templett1[tt, 1] = (ILoctemp[3, 1] + ILoctemp[0, 1]) / 2;
            templett1[tt, 0] = ILoctemp[6, 0];
            //Curvas de nivel superior e inferior
            //Variable que indicará las curvas de nivel que se realizarán
            int t = 1;
            //Variable que indicará los puntos definidos para poder representar
las curvas de nivel
            int i = 0;
            //Definimos 2 matrices temoorales 1 para la parte de arriba y otra
para la parte de abajo.
            double[,] tar = new double[100, 2];
            double[,] tab = new double[100, 2];
            //Matrices temporal que contendrá las coordenadas para los
letreros(tendrán que ser girados)
            double[,] templet = new double[100, 2];
            double[,] templett = new double[100, 2];
            for (double tempz = this.tempzs[1]; tad[i, 1] < ILoctemp[11, 1];
tempz = tempz + 5)
            {
                this.tempzs[t] = tempz;
                tar[i, 1] = tad[i, 1];
                tab[i, 1] = tad[i + 1, 1];
                tar[i, 0] = ILoctemp[3, 0] + (tar[i, 1] - ILoctemp[3, 1]) /
(Math.Tan(30 * 2 * Math.PI / 360));
                tab[i, 0] = ILoctemp[0, 0] + Math.Abs(tab[i, 1] - ILoctemp[0, 1])
/ (Math.Tan(30 * 2 * Math.PI / 360));
                templet[t, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
                templet[t, 1] = tar[i, 1];
                templett[t, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
                templett[t, 1] = tab[i, 1];
                i = i + 1;
                tar[i, 1] = tad[i - 1, 1];
                tab[i, 1] = tad[i, 1];
                tar[i, 0] = ILoctemp[2, 0] - (tar[i, 1] - ILoctemp[2, 1]) /
(Math.Tan(30 * 2 * Math.PI / 360));
                tab[i, 0] = ILoctemp[1, 0] - Math.Abs(tab[i, 1] - ILoctemp[1, 1])
/ (Math.Tan(30 * 2 * Math.PI / 360));
                i = i + 1;
                t = t + 1;
            }
            templet[0, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
            templet[0, 1] = ILoctemp[2, 1];
            templett[0, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
            templett[0, 1] = ILoctemp[1, 1];
            templet[t, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
```

```
            templet[t, 1] = ILoctemp[11, 1];
            templett[t, 0] = (ILoctemp[0, 0] + ILoctemp[1, 0]) / 2;
            templett[t, 1] = ILoctemp[9, 1];
            int h = 12;
            for (int k = 0; k < i; k++)
            {
                ILoctemp[h, 0] = tar[k, 0];
                ILoctemp[h, 1] = tar[k, 1];
                h = h + 1;
            }
            for (int k = 0; k < i; k++)
            {
                ILoctemp[h, 0] = tab[k, 0];
                ILoctemp[h, 1] = tab[k, 1];
                h = h + 1;
            }
            for (int k = 0; k < j; k++)
            {
                ILoctemp[h, 0] = tad[k, 0];
                ILoctemp[h, 1] = tad[k, 1];
                h = h + 1;
            }
            for (int k = 0; k < j; k++)
            {
                ILoctemp[h, 0] = tai[k, 0];
                ILoctemp[h, 1] = tai[k, 1];
                h = h + 1;
            }
            //Ahora lo pasamos todo a la matriz principal que contendrá todos los
puntos con la orientación original.
            double[,] ILoc = new double[h, 2];
            for (int k = 0; k < h; k++)
            {
                double[] girILoc = new double[2];
                girILoc = its.giroinv(ILoctemp[k, 0], ILoctemp[k, 1], ang);
                ILoc[k, 0] = (girILoc[0]) * this.escala[0] / this.escala[1];
                ILoc[k, 1] = (girILoc[1]) * this.escala[0] / this.escala[1];
            }
            int zt = 0;
            for (int k = 0; k < tt; k++)
            {
                zstemp[zt, 0] = tempzs[k];
                zstemp[zt, 1] = templet1[k, 0];
                zstemp[zt, 2] = templet1[k, 1];
                zt = zt + 1;
            }
            for (int k = 0; k < tt; k++)
            {
                zstemp[zt, 0] = tempzs[k];
                zstemp[zt, 1] = templett1[k, 0];
                zstemp[zt, 2] = templett1[k, 1];
                zt = zt + 1;
            }
            for (int k = 0; k < t; k++)
            {
                zstemp[zt, 0] = tempzs[k];
                zstemp[zt, 1] = templet[k, 0];
                zstemp[zt, 2] = templet[k, 1];
                zt = zt + 1;
            }
            for (int k = 0; k < t; k++)
            {
```

```
                zstemp[zt, 0] = tempzs[k];
                zstemp[zt, 1] = templett[k, 0];
                zstemp[zt, 2] = templett[k, 1];
                zt = zt + 1;
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[zt, 3];
            for (int k = 0; k < zt; k++)
            {
                this.zs[k, 0] = zstemp[k, 0];
                double[] girzs = new double[2];
                girzs = its.giroinv(zstemp[k, 1], zstemp[k, 2], ang);
                this.zs[k, 1] = (girzs[0]) * this.escala[0] / this.escala[1];
                this.zs[k, 2] = (girzs[1]) * this.escala[0] / this.escala[1];
            }
            return ILoc;
        }

        //Método que nos devolverá un vector con 4 variables de tipo double,
        // los 2 primeros variables indicará la posición x e y de la instalación
radioeléctrica
        // las otras 2 variables indicarán el radio de las 2 circuferencias que
formarán
        //la zona de seguridad y la zona de limitación de alturas
        public double[] GetSRadE()
        {
            //Matriz temporal que contendrá los puntos base de la SUPERFICIE
RADIOELÉCTRICA
            double[] Sradtemp = new double[300];
            string z = Convert.ToString(Math.Round(radpos[2], 0));
            double zz = Convert.ToDouble(z.Substring(z.Length - 1, 1));
            this.tempzs[0] = Math.Round(radpos[2]);
            int t = 1;
            int i = 4;
            if (Math.Round(radpos[2]) % 5 == 0)
            {
                this.tempzs[1] = this.tempzs[0] + 5;
            }
            else
            {
                if (zz < 5)
                {
                    this.tempzs[1] = this.tempzs[0] + (5 - zz);
                }
                else
                {
                    this.tempzs[1] = this.tempzs[0] + (10 - zz);
                }
            }
            double[,] zstemp = new double[100, 3];
            if (this.srad.Getpen() != 100)
            {
                for (double tempz = this.tempzs[1]; tempz < (this.srad.Getzalt()
* this.srad.Getpen()) / 100 + radpos[2] && Sradtemp[i - 1] < this.srad.Getzalt();
tempz = tempz + 5)
                {
                    this.tempzs[t] = tempz;
                    Sradtemp[i] = this.srad.Getzseg() + (this.tempzs[t] -
this.tempzs[0]) * 100 / this.srad.Getpen();
                    zstemp[t, 0] = this.tempzs[t];
                    zstemp[t, 1] = this.radpos[0];
                    zstemp[t, 2] = Sradtemp[i];
```

```
                    i = i + 1;
                    t = t + 1;
                }
            }
            double[] SRadI = new double[i];
            SRadI[0] = this.radpos[0] * this.escala[0] / this.escala[1];
            SRadI[1] = this.radpos[1] * this.escala[0] / this.escala[1];
            SRadI[2] = this.srad.Getzseg() * this.escala[0] / this.escala[1];
            SRadI[3] = (this.srad.Getzalt()) * this.escala[0] / this.escala[1];
            zstemp[0, 0] = this.tempzs[0];
            zstemp[0, 1] = radpos[0];
            zstemp[0, 2] = SRadI[2];

            for (int j = 4; j < i; j++)
            {
                SRadI[j] = Sradtemp[j];
            }
            //Trasladamos las coordenadas de la matriz zs
            this.zs = new double[t, 3];
            for (int j = 0; j < t; j++)
            {
                this.zs[j, 0] = zstemp[j, 0];
                double[] girzs = new double[2];
                this.zs[j, 1] = zstemp[j, 1] * this.escala[0] / this.escala[1];
                this.zs[j, 2] = zstemp[j, 2] * this.escala[0] / this.escala[1];
            }
            return SRadI;
        }
        public double[,] Gettext()
        {
            return this.zs;
        }

    }
}
```

Intersecciones.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LibreriaDimPen;

namespace LibreriaIntersecciones
{
    public class Intersecciones
    {
        //Dado la posición en 3D de un objeto se determinará si éste representa
un obstáculo o no para las diferentes superfícies limitadoras.
        //Definimos las variables que contendran el rango de las x's y las y'. Si
la z del obstáculo es igual o superior a alguna z del anterior
        // se considerará que hay una intersección.
        double xmax;
        double xmin;
        double ymax;
        double ymin;
        //Variable que indicará la altura a la que s eproduce la intersección.
        double alt;

        //Variables secundarias que nos servirán para determinar diversos puntos
de interés de la superficie de aproximación y de la ascd
        double xmax2;
        double xmin2;
        double ymax2;
        double ymin2;
        double xx2;//también se usará en ils/gp
        //Variables para los límites de la aproximación interna
        double xmax3;
        double xmin3;
        double ymax3;
        double ymin3;

        //Variables a partir de las cuales definiremos los puntos límite de la
superficie del ils/loc
        double x1;
        double x2;
        double y1;
        double y2;
        //Variables a partir de las cuales definiremos los puntos límite de la
superficie del ils/gp
        double xx1;
        double yy1;
        double yy2;

        //y de rectas, pendientes y constantes de IGp//ILSLoc
        double yar;
        double yab;
        double m1;
        double m2;
        double b1;
        double b2;
        double x11;
        double x22;
        // Constructor vacío
        public Intersecciones()
        {
            this.xmax = 0;
```

```
            this.xmin = 0;
            this.ymax = 0;
            this.ymin = 0;
            this.alt = 0;
            this.xmax2 = 0;
            this.xmin2 = 0;
            this.ymax2 = 0;
            this.ymin2 = 0;
            this.xx2 = 0;
            this.xmax3 = 0;
            this.xmin3 = 0;
            this.ymax3 = 0;
            this.ymin3 = 0;
        }
        //Método que nos devolverá la altura donde se produce la intersección.
        public double Getalt()
        {
            return this.alt;
        }
        //Método que nos devolverá un vector con los límites de las superfícies
        public double[] Getlim()
        {
            double[] lim = { this.xmax, this.xmin, this.ymax, this.ymin };
            return lim;
        }
        //Método que nos devolverá un vector con los límites secundario de la
superfície de aproximación para operaciones de precisión
        public double[] Getlim2()
        {
            double[] lim2 = { this.xmax2, this.xmin2, this.ymax2, this.ymin2 };
            return lim2;
        }
        //Método que nos devolverá un vector con los límites secundario de la
superfície de aproximación interna
        public double[] Getlim3()
        {
            double[] lim3 = { this.xmax3, this.xmin3, this.ymax3, this.ymin3 };
            return lim3;
        }

        //Método que nos devolverá la coordenada x dónde la superficie de ascenso
varía su pendiente para pistas de clave 3 o 4.
        public double [] Getxxs()
        {
            double[] xxs = { this.xx1, this.xx2, this.yy1, this.yy2 };
            return xxs;
        }

        //Método que nos devolverá las coordenadas de la zona de seguridad de
ils/loc
        public double [] Getxyils()
        {
            double[] xyils = { this.x1, this.x2, this.y1, this.y2 };
            return xyils;
        }
        //Método que nos devuelve las ys, las pendientes y las constantes  de la
srectas del igp
        public double[] Getya()
        {
            double[] ya = { this.yar, this.yab, this.m1, this.m2, this.b1,
this.b2, this.x11, this.x22 };
            return ya;
```

```
        }
        //Para simplificar los cálculos se transladarán (girarán) los puntos un
ángulo 90º-(orientación de la pista respecto el norte magnético)
        // en sentido de las agujas del reloj.
        public double[] giro(double[] pos, double ang)
        {
            double x = pos[0] * Math.Cos(((ang) *2* Math.PI) / 360) - pos[1] *
Math.Sin(((ang) *2* Math.PI) / 360);
            double y = pos[1] * Math.Cos(((ang) *2* Math.PI) / 360) + pos[0] *
Math.Sin(((ang) *2* Math.PI) / 360);
            double [] giropos={x,y,pos[2]};
            return giropos;
        }
        //Método que realiza la translación inversa, un punto que es transladado
por ambos métodos volvería a su posición inicial.
        public double[] giroinv(double x, double y, double ang)
        {
            double xx = y * Math.Sin(((ang) * 2 * Math.PI) / 360) + x *
Math.Cos(((ang) * 2 * Math.PI) / 360);
            double yy = y * Math.Cos(((ang) * 2 * Math.PI) / 360) - x *
Math.Sin(((ang) * 2 * Math.PI) / 360);
            double[] giroinv = { xx, yy };
            return giroinv;
        }
        //Horizontal Interna
        public bool inthoint(double longitudpista, double radio, double [] pos,
double [] pistapos)
        {
            bool inthi = false;
            ymax = radio+pistapos[1];
            ymin = -radio + pistapos[1];
            xmax = pistapos[0] + (longitudpista / 2) + radio;
            xmin = pistapos[0] - (longitudpista / 2) - radio;
            if (pos[2] >= pistapos[2]+ 45)
            {
                if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= xmin &&
pos[0] <= xmax))
                {
                    if (pos[0] >= (xmin + radio) && pos[0] <= (xmax - radio))
                    {
                        inthi = true;
                    }
                    else
                    {
                        if (pos[0] >= xmin && pos[0] <= (xmin + radio))
                        {
                            double x1 = Math.Sqrt(Math.Pow(radio, 2) -
Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] - longitudpista / 2;
                            double x2 = -Math.Sqrt(Math.Pow(radio, 2) -
Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] - longitudpista / 2;
                            if (pos[0] < 0)
                            {
                                if (pos[0] >= x2)
                                {
                                    inthi = true;
                                }
                            }
                            else
                            {
                                if (pos[0] >= x1)
                                {
                                    inthi = true;
```

```
                            }
                        }
                    }
                    if (pos[0] >= (xmax - radio) && pos[0] <= xmax)
                    {
                        double x1 = Math.Sqrt(Math.Pow(radio, 2) -
Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] + longitudpista / 2;
                        double x2 = -Math.Sqrt(Math.Pow(radio, 2) -
Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] + longitudpista / 2;
                        if (pos[0] < 0)
                        {
                            if (pos[0] <= x2)
                            {
                                inthi = true;
                            }
                        }
                        else
                        {
                            if (pos[0] <= x1)
                            {
                                inthi = true;
                            }
                        }
                    }
                }
            }
        }
        this.alt = 45;
        return inthi;
    }

    //Cónica
    public bool intconi(double longitudpista, double radio, double [] pos,
double [] pistapos,double pendiente, double altura)
    {
        bool intcon = false;
        //Definimos una nueva variable llamada radiopen la cual va variando
dependiendo de la x e la y, proporcionalmente con la pendiente
        //de la superfície cónica (es decir el valor del radio en la posición
del obstáculo si éste está dentro del rango de la superfície cónica) .
        double radiopen;
        double radiomax = radio + (altura / (pendiente / 100));
        if (pos[2]>=45)
        {
            xmax = pistapos[0] + longitudpista / 2 + radiomax;
            xmin = pistapos[0] - longitudpista / 2 - radiomax;
            ymax = pistapos[1] + radiomax;
            ymin = pistapos[1] - radiomax;
            if ((pos[0] >= xmin && pos[0] <= xmax) && (pos[1] >= ymin &&
pos[1] <= ymax))
            {
                if (pos[0] >= (pistapos[0] - longitudpista / 2) && pos[0] <=
(pistapos[0] + longitudpista / 2))
                {
                    if (pos[1] >= (pistapos[1] + radio))
                    {
                        double z = (pendiente / 100) * (pos[1] - (pistapos[1]
+ radio)) + 45;
                        if (pos[2] >= z)
                        {
                            intcon = true;
                            this.alt = z;
```

```
                            }
                        }
                        else
                        {
                            if (pos[1] <= (pistapos[1] - radio))
                            {
                                double z = (-pendiente / 100) * (pos[1] -
(pistapos[1] - radio)) + 45;
                                if (pos[2] >= z)
                                {
                                    intcon = true;
                                    this.alt = z;
                                }
                            }
                        }
                    }
                    else
                    {
                        if (pos[0] >= (pistapos[0] + longitudpista / 2) && pos[0]
<= xmax)
                        {
                            radiopen = -radio + Math.Sqrt(Math.Pow((pos[0] -
(pistapos[0] + longitudpista / 2)), 2) + Math.Pow((pos[1] - pistapos[1]), 2));
                            if ((radiopen + radio) <= radiomax)
                            {
                                double z = (pendiente / 100) * radiopen + 45;
                                if (pos[2] >= z)
                                {
                                    if (pos[1] <= (pistapos[1] - radio) || pos[1]
>= (pistapos[1] + radio))
                                    {
                                        intcon = true;
                                        this.alt = z;
                                    }
                                    else
                                    {
                                        double x1 = Math.Sqrt(Math.Pow(radio, 2)
- Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] + longitudpista / 2;
                                        double x2 = -Math.Sqrt(Math.Pow(radio, 2)
- Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] + longitudpista / 2;
                                        if (pos[0] < 0)
                                        {
                                            if (pos[0] >= x2)
                                            {
                                                intcon = true;
                                                this.alt = z;
                                            }
                                        }
                                        else
                                        {
                                            if (pos[0] >= x1)
                                            {
                                                intcon = true;
                                                this.alt = z;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                        else
                        {
```

```
                            radiopen = -radio + Math.Sqrt(Math.Pow((pos[0] -
(pistapos[0] - longitudpista / 2)), 2) + Math.Pow((pos[1] - pistapos[1]), 2));
                            if ((radiopen + radio) <= radiomax)
                            {
                                double z = (pendiente / 100) * radiopen + 45;
                                if (pos[2] >= z)
                                {
                                    if (pos[1] <= (pistapos[1] - radio) || pos[1]
>= (pistapos[1] + radio))
                                    {
                                        intcon = true;
                                        this.alt = z;
                                    }
                                    else
                                    {
                                        double x1 = Math.Sqrt(Math.Pow(radio, 2)
- Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] - longitudpista / 2;
                                        double x2 = -Math.Sqrt(Math.Pow(radio, 2)
- Math.Pow((pos[1] - (pistapos[1])), 2)) + pistapos[0] - longitudpista / 2;
                                        if (pos[0] < 0)
                                        {
                                            if (pos[0] <= x2)
                                            {
                                                intcon = true;
                                                this.alt = z;
                                            }
                                        }
                                        else
                                        {
                                            if (pos[0] <= x1)
                                            {
                                                intcon = true;
                                                this.alt = z;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            return intcon;
        }

        //Aproximación y Aproximación Interna
        public bool[] intaprox(string aprox, DimPen dp, double [] pos, double []
pistapos, int clave, double longitudpista, double posumbral, double sentido)
        {
            bool [] intap={false,false};
            //Variable local que se utilizará posteriormente para determinar el
valor de x en el inicio de la sup.
            double xx;
            if (sentido == -1)
            {
                xmax = pistapos[0] - longitudpista / 2 - dp.Getdistuap() +
posumbral;
                xmin = xmax - dp.Getpseclong();
                xx = xmax;
            }
            else
            {
```

```
                xmin = pistapos[0] + longitudpista / 2 + dp.Getdistuap() -
posumbral;
                xmax = xmin + dp.Getpseclong();
                xx = xmin;
            }
            ymax = pistapos[1] + dp.Getdivap() / 100 * dp.Getpseclong() +
dp.Getlongbiap() / 2;
            ymin = pistapos[1] - dp.Getdivap() / 100 * dp.Getpseclong() -
dp.Getlongbiap() / 2;
            if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= xmin && pos[0]
<= xmax))
            {
                //Variable que indicará el valor de z en la posición del
obstáculo
                double z = (Math.Abs(pos[0] - xx) * (dp.Getpsecpen() / 100)) +
pistapos[2];
                if (pos[2] >= z)
                {
                    if ((pos[1] >= (ymin + dp.Getdivap() / 100 *
dp.Getpseclong()) && pos[1] <= (ymax - dp.Getdivap() / 100 * dp.Getpseclong())))
                    {
                        intap[0] = true;
                        this.alt = z;
                    }
                    else
                    {
                        if ((pos[1] >= (pistapos[1] + dp.Getlongbiap() / 2) &&
(pos[1] <= ymax)))
                        {
                            double y0 = (dp.Getdivap() / 100 * Math.Abs(pos[0] -
xx)) + (pistapos[1] + dp.Getlongbiap() / 2);
                            if (pos[1] <= y0)
                            {
                                intap[0] = true;
                                this.alt = z;
                            }
                        }
                        else
                        {
                            double y0 = (- dp.Getdivap() / 100 * Math.Abs(pos[0]
- xx)) + (pistapos[1] - dp.Getlongbiap() / 2);
                            if (pos[1] >= y0)
                            {
                                intap[0] = true;
                                this.alt = z;
                            }
                        }
                    }
                }
            }
            if (aprox == "Prec1" || aprox == "Prec23" || (aprox == "NoPrec" &&
(clave == 3 || clave == 4)))
            {
                if (intap[0] == false)
                {
                    double x1;
                    double x2;
                    double y1 = ymax;
                    double y2 = ymin;
                    double xx2;
                    if (sentido == -1)
                    {
```

```
                        xmax2 = xmin;
                        xmin2 = xmax2 - dp.Getsseclong() - dp.Getshorlong();
                        xx = xmax2;
                        x1 = xmin2 + dp.Getshorlong();
                        x2 = xmax2;
                        xx2 = x1;
                    }
                    else
                    {
                        xmin2 = xmax;
                        xmax2 = xmin2 + dp.Getsseclong() + dp.Getshorlong();
                        xx = xmin2;
                        x1 = xmin2;
                        x2 = xmax2 - dp.Getshorlong();
                        xx2 = x2;
                    }
                    ymin2 = ymin - ((dp.Getsseclong() + dp.Getshorlong())/100) *
dp.Getdivap();
                    ymax2 = ymax + ((dp.Getsseclong() + dp.Getshorlong())/100) *
dp.Getdivap();
                    if ((pos[1] >= ymin2 && pos[1] <= ymax2) && (pos[0] >= xmin2
&& pos[0] <= xmax2))
                    {
                        if ((pos[0] >= x1 && pos[0] <= x2) && ((pos[1] >= (ymin2
+ dp.Getshorlong()* dp.Getdivap()/100)
                            && (pos[1] <= (ymax2 - dp.Getshorlong() *
dp.Getdivap()/100)))))
                        {
                            double z = (Math.Abs(pos[0] - xx) * (dp.Getssecpen())
/ 100) + (pistapos[2] + (dp.Getpsecpen() / 100) * dp.Getpseclong());
                            if (pos[2] >= z)
                            {
                                if (pos[1] <= y1 && pos[1] >= y2)
                                {
                                    intap[0] = true;
                                    this.alt = z;
                                }
                                else
                                {
                                    if (pos[1] >= (y1 + (dp.Getsseclong() *
dp.Getdivap()) / 100))
                                    {
                                        double y0 = (Math.Abs(pos[0] - xx) *
dp.Getdivap() / 100) + y1;
                                        if (pos[1] <= y0)
                                        {
                                            intap[0] = true;
                                            this.alt = z;
                                        }
                                    }
                                    else
                                    {
                                        double y0 = (Math.Abs(pos[0] - xx) * (-
dp.Getdivap() / 100)) + y2;
                                        if (pos[1] >= y0)
                                        {
                                            intap[0] = true;
                                            this.alt = z;
                                        }
                                    }
                                }
                            }
```

```
                        }
                        else
                        {
                            if ((pos[0] <= xx2 && pos[0] >= xmin2 && sentido ==-
1) || (pos[0] <= xmax2 && pos[0] >= xx2 && sentido==1))
                            {
                                double z = dp.Getssecalt();
                                if (pos[2] > z)
                                {
                                    if ((pos[1] <= (ymax2 - ((dp.Getshorlong() *
dp.Getdivap()) / 100))) &&
                                        (pos[1] >= (ymin2 + ((dp.Getshorlong() *
dp.Getdivap()) / 100))))
                                    {
                                        intap[0] = true;
                                        this.alt = z;
                                    }
                                    else
                                    {
                                        if (pos[1] <= ymax2 && pos[1] >= (ymax2 -
((dp.Getshorlong() * dp.Getdivap()) / 100)))
                                        {
                                            double y0 = (Math.Abs(pos[0] - xx2) *
(dp.Getdivap() / 100)) + (ymax2 - ((dp.Getshorlong() * dp.Getdivap()) / 100));
                                            if (pos[1] <= y0)
                                            {
                                                intap[0] = true;
                                                this.alt = z;
                                            }
                                        }
                                        else
                                        {
                                            double y0 = (Math.Abs(pos[0] - xx2) *
(-dp.Getdivap() / 100)) + (ymin2 + ((dp.Getshorlong() * dp.Getdivap()) / 100));
                                            if (pos[1] >= y0)
                                            {
                                                intap[0] = true;
                                                this.alt = z;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                    if (aprox == "Prec1" || aprox == "Prec23")
                    {
                        if (sentido == -1)
                        {
                            xmax3 = pistapos[0] - longitudpista / 2 -
dp.Getdistuaint() + posumbral;
                            xmin3 = xmax3 - dp.Getlongaint();
                            xx = xmax3;
                        }
                        else
                        {
                            xmin3 = pistapos[0] + longitudpista / 2 +
dp.Getdistuaint() - posumbral;
                            xmax3 = xmin3 + dp.Getlongaint();
                            xx = xmin3;
                        }
```

```
                    ymax3 = pistapos[1] + dp.Getanchapint() / 2;
                    ymin3 = pistapos[1] - dp.Getanchapint() / 2;
                    if ((pos[1] >= ymin3 && pos[1] <= ymax3) && (pos[0] >= xmin3
&& pos[0] <= xmax3))
                    {
                        double z = Math.Abs(pos[0] - xx) * (dp.Getpenaint() /
100) + pistapos[2];
                        if (pos[2] >= z)
                        {
                            intap[1] = true;
                            this.alt = z;
                        }
                    }
                }
            }
            return intap;
        }

        //Transición y Transición Interna
        public bool[] inttrans(DimPen dp, Franjadim fjd, double [] pistapos,
double [] pos, double sentido,double longitudpista, string aprox,double
posumbral)
        {
            bool[] inTrans = { false, false };
            ymax = pistapos[1] + (fjd.Getfrananch() + (45 * 100 /
dp.Getpentrans()));
            ymin = pistapos[1] - (fjd.Getfrananch() + (45 * 100 /
dp.Getpentrans()));
            double x1;
            double x2;
            if (sentido == -1)
            {
                xmin = (-100 / dp.Getdivap()) * (ymax - (pistapos[1] +
dp.Getlongbiap() / 2)) + (pistapos[0] - longitudpista / 2 - dp.Getdistuap() +
posumbral);
                xmax = pistapos[0] + (longitudpista / 2 + fjd.Getfranlong());
                x1 = xmin;
                x2 = (pistapos[0] - longitudpista / 2 - dp.Getdistuap() +
posumbral);
            }
            else
            {
                xmin = pistapos[0] - (longitudpista / 2 + fjd.Getfranlong());
                xmax = (100 / dp.Getdivap()) * (ymax - (pistapos[1] +
dp.Getlongbiap() / 2)) + (pistapos[0] + longitudpista / 2 + dp.Getdistuap() -
posumbral);
                x1 = (pistapos[0] + longitudpista / 2 + dp.Getdistuap() -
posumbral);
                x2 = xmax;
            }
            if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= xmin && pos[0]
<= xmax))
            {
                if (pos[0] >= x1 && pos[0] <= x2)
                {
                    if (pos[1] <= ymax && pos[1] >= (pistapos[1] +
fjd.Getfrananch()))
                    {
                        double z = Math.Abs(pos[1] - (pistapos[1] +
fjd.Getfrananch())) * (dp.Getpentrans() / 100) + pistapos[2];
                        if (pos[2] >= z)
                        {
```

```
                            double y0 = Math.Abs(pos[0] - x1) * (dp.Getdivap() /
100) + (pistapos[1] + fjd.Getfrananch());
                            if (pos[1] >= y0)
                            {
                                inTrans[0] = true;
                                this.alt = z;
                            }
                        }
                    }
                    if (pos[1] >= ymin && pos[1] <= (pistapos[1] -
fjd.Getfrananch()))
                    {
                        double z = Math.Abs(pos[1] - (pistapos[1] -
fjd.Getfrananch())) * (dp.Getpentrans() / 100) + pistapos[2];
                        if (pos[2] >= z)
                        {
                            double y0 = Math.Abs(pos[0] - x1) * (-dp.Getdivap() /
100) + (pistapos[1] + fjd.Getfrananch());
                            if (pos[1] <= y0)
                            {
                                inTrans[0] = true;
                                this.alt = z;
                            }
                        }
                    }
                }
                else
                {
                    if (pos[1] <= ymax && pos[1] >= (pistapos[1] +
fjd.Getfrananch()))
                    {
                        double z = Math.Abs(pos[1] - (pistapos[1] +
fjd.Getfrananch())) * (dp.Getpentrans() / 100) + pistapos[2];
                        if (pos[2] >= z)
                        {
                            inTrans[0] = true;
                            this.alt = z;
                        }
                    }
                    if (pos[1] >= ymin && pos[1] <= (pistapos[1] -
fjd.Getfrananch()))
                    {
                        double z = Math.Abs(pos[1] - (pistapos[1] -
fjd.Getfrananch())) * (dp.Getpentrans() / 100) + pistapos[2];
                        if (pos[2] >= z)
                        {
                            inTrans[0] = true;
                            this.alt = z;
                        }
                    }
                }
            }
            if(aprox=="Prec1" || aprox=="Prec23")
            {
                //Variable que indicará la longitud necesaria, a partir del
pendiente, con la que la superficie de aterrizaje interrumpido alcanzará los 45m.
                double longsai = ((45 * 100) / dp.Getpensai());
                //Variable que indicará la posición a la cual se situará el
inicio de la superficie de aterrizaje interrumpido.
                double possai;
                if (aprox == "Prec1")
                {
```

```
                    possai = dp.Getdistusai() - longitudpista / 2;
                }
                else
                {
                    //Si la longitud de la pista ,teniendo en cuenta la posición
del umbral, es menor que la distancia de la superficie desde el umbral. La SAI se
situará a
                    // a partir del extremo de la pista.
                    if ((posumbral + dp.Getdistusai() > longitudpista))
                    {
                        possai = longitudpista / 2;
                    }
                    else
                    {
                        possai = longitudpista / 2 - (longitudpista - posumbral -
dp.Getdistusai());
                    }
                }
                if (sentido == -1)
                {
                    xx1 = pistapos[0] - longitudpista / 2 - dp.Getdistuaint() +
posumbral;
                    xmin3 = xx1 - dp.Getlongaint();
                    xx2 = pistapos[0] + possai;
                    xmax3 = xx2 + longsai;
                }
                else
                {
                    xx1 = pistapos[0] + longitudpista / 2 + dp.Getdistuaint() -
posumbral;
                    xmax3 = xx1 + dp.Getlongaint();
                    xx2 = pistapos[0] - possai;
                    xmin3 = xx2 - longsai;
                }
                ymax3 = pistapos[1] + (dp.Getanchapint()/2 + (45 * 100 /
dp.Getpentransint()));
                ymin3 = pistapos[1] - (dp.Getanchapint()/2 + (45 * 100 /
dp.Getpentransint()));
                if ((pos[0] >= xmin3 && pos[0] <= xmax3) && (pos[1] >= ymin3 &&
pos[1] <= ymax3))
                {
                    if ((pos[0] >= xx1 && pos[0] <= xx2) || (pos[0] <= xx1 &&
pos[0] >= xx2))
                    {
                        if (pos[1] >= pistapos[1] + dp.Getanchapint()/2)
                        {
                            double z = Math.Abs(pos[1] - (pistapos[1] +
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                            if (pos[2] >= z)
                            {
                                inTrans[1] = true;
                                this.alt = z;
                            }
                        }
                        if (pos[1] <= pistapos[1] - dp.Getanchapint()/2)
                        {
                            double z = Math.Abs(pos[1] - (pistapos[1] -
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                            if (pos[2] >= z)
                            {
                                inTrans[0] = true;
                                this.alt = z;
```

```
                            }
                        }
                    }
                    else
                    {
                        if ((pos[0] <= xx1 && (pos[0] >= xmin3) || (pos[0] >= xx1
&& pos[0] <= xmax3)))
                        {
                            if (pos[1] >= pistapos[1] + dp.Getanchapint()/2)
                            {
                                double z = Math.Abs(pos[1] - (pistapos[1] +
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                                if (pos[2] >= z)
                                {
                                    double y0 = Math.Abs(pos[0] - xx1) * (-
dp.Getdivsai() / 100) + (pistapos[1] + dp.Getanchapint()/2);
                                    if (pos[1] <= y0)
                                    {
                                        inTrans[1] = true;
                                        this.alt = z;
                                    }
                                }
                            }
                            if (pos[1] <= pistapos[1] - dp.Getanchapint()/2)
                            {
                                double z = Math.Abs(pos[1] - (pistapos[1] +
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                                if (pos[2] >= z)
                                {
                                    double y0 = Math.Abs(pos[0] - xx1) *
(dp.Getdivsai() / 100) + (pistapos[1] + dp.Getanchapint()/2);
                                    if (pos[1] >= y0)
                                    {
                                        inTrans[1] = true;
                                        this.alt = z;
                                    }
                                }
                            }
                        }
                        if ((pos[0] >= xx2 && pos[0] <= xmax3) || (pos[0] >=
xmin3 && pos[0] <= xx2))
                        {
                            if (pos[1] >= pistapos[1] + dp.Getanchapint()/2)
                            {
                                double z = Math.Abs(pos[1] - (pistapos[1] +
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                                if (pos[2] >= z)
                                {
                                    double y0 = Math.Abs(pos[0] - xx2) *
(dp.Getdivsai() / 100) + (pistapos[1] + dp.Getanchapint()/2);
                                    if (pos[1] >= y0)
                                    {
                                        inTrans[1] = true;
                                        this.alt = z;
                                    }
                                }
                            }
                            if (pos[1] <= pistapos[1] - dp.Getanchapint()/2)
                            {
                                double z = Math.Abs(pos[1] - (pistapos[1] +
dp.Getanchapint()/2)) * (dp.Getpentransint() / 100) + pistapos[2];
                                if (pos[2] >= z)
```

```
                                {
                                    double y0 = Math.Abs(pos[0] - xx2) * (-
dp.Getdivsai() / 100) + (pistapos[1] + dp.Getanchapint()/2);
                                    if (pos[1] <= y0)
                                    {
                                        inTrans[1] = true;
                                        this.alt = z;
                                    }
                                }
                            }
                        }
                    }
                }
            }
            return inTrans;
        }

        //Superficie de Aterrizaje Interrumpido
        public bool intsai(DimPen dp, double [] pos, double [] pistapos, double
longitudpista, double sentido, double posumbral, string aprox)
        {
            bool intSAI = false;
            //Sabemos que el borde exterior estará a una altura de 45m(h de
H.interna), a partir de este dato y conociendo el valor de la pendiente de la
superficie
            //podremos calcular la extensión longitudinal de la superficie de
aproximación interrumpida
            double longsai=((45*100)/dp.Getpensai());
            double xx;
            double possai;
            if (aprox == "Prec1")
            {
                possai = dp.Getdistusai() - longitudpista / 2;
            }
            else
            {
                //Si la longitud de la pista ,teniendo en cuenta la posición del
umbral, es menor que la distancia de la superficie desde el umbral. La SAI se
situará a
                // a partir del extremo de la pista.
                if ((posumbral + dp.Getdistusai() > longitudpista))
                {
                    possai = longitudpista / 2;
                }
                else
                {
                    possai = longitudpista / 2 - (longitudpista - posumbral -
dp.Getdistusai());
                }
            }
            if (sentido == -1)
            {
                xmin = pistapos[0] + possai;
                xmax = xmin + longsai;
                xx = xmin;
            }
            else
            {
                xmax = pistapos[0] - possai;
                xmin = xmax - longsai;
                xx = xmax;
            }
```

```
            ymax = pistapos[1] + dp.Getlongbisai() / 2 + longsai *
(dp.Getdivsai()) / 100;
            ymin = pistapos[1] - dp.Getlongbisai() / 2 - longsai *
(dp.Getdivsai()) / 100;
            if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= xmin && pos[0]
<= xmax))
            {
                double z = Math.Abs(pos[0] - xx) * (dp.Getpensai() / 100) +
pistapos[2];
                if (pos[2] >= z)
                {
                    if (pos[1] <= (ymax - (longsai * (dp.Getdivsai()) / 100)) &&
                        pos[1] >= (ymin + (longsai * (dp.Getdivsai()) / 100)))
                    {
                        intSAI = true;
                        this.alt = z;
                    }
                    else
                    {
                        if (pos[1] >= (pistapos[1] + dp.Getlongbisai() / 2) &&
pistapos[1] <= ymax)
                        {
                            double y1 = Math.Abs(pos[0] - xx) * dp.Getdivsai() /
100 + (pistapos[1] + dp.Getlongbisai() / 2);
                            if (pos[1] <= y1)
                            {
                                intSAI = true;
                                this.alt = z;
                            }
                        }
                        else
                        {
                            double y1 = Math.Abs(pos[0] - xx) * (-dp.Getdivsai())
/ 100 + (pistapos[1] + dp.Getlongbisai() / 2);
                            if (pos[1] >= y1)
                            {
                                intSAI = true;
                                this.alt = z;
                            }
                        }
                    }
                }
            }
            return intSAI;
        }

        //DE ASCENSO EN EL DESPEGUE
        public bool intasc(DimPen dp, double [] pos, double [] pistapos, double
longitudpista, double sentido)
        {
            bool inTasc = false;
            // sentido de despegue
            double xx1;
            bool clave34 = false;
            if (sentido == 1)
            {
                xmin = pistapos[0] + dp.Getdistextpascd() + longitudpista / 2;
                xx1 = xmin;
                xmax = xmin + dp.Getlongascd();
                if (longitudpista >= 1200)
                {
```

```
                    xx2 = xmin + ((dp.Getanchascd() - dp.Getlongbiascd()) * 100 /
(2 * dp.Getdivascd()));
                    clave34 = true;
                }
                else
                {
                    xx2 = xmax;
                }
            }
            else
            {
                xmax = pistapos[0] - dp.Getdistextpascd() - longitudpista / 2;
                xx1 = xmax;
                xmin = xmax +- dp.Getlongascd();
                if (longitudpista >= 1200)
                {
                    xx2 = xmax - ((dp.Getanchascd() - dp.Getlongbiascd()) * 100 /
(2 * dp.Getdivascd()));
                    clave34 = true;
                }
                else
                {
                    xx2 = xmin;
                }
            }
            ymax = pistapos[1] + (dp.Getanchascd() / 2);
            ymin = pistapos[1] - (dp.Getanchascd() / 2);
            if ((pos[0] >= xmin && pos[0] <= xmax) && (pos[1] >= ymin && pos[1]
<= ymax))
            {
                if ((clave34 == true))
                {
                    if ((pos[0] >= xx1 && pos[0] <= xx2) || (pos[0] >= xx2 &&
pos[0] <= xx1))
                    {
                        double z = Math.Abs(pos[0] - xx1) * dp.Getdivascd() / 100
+ pistapos[2];
                        if (pos[2] >= z)
                        {
                            if (pos[1] >= (pistapos[1] - dp.Getlongbiascd() / 2)
&& pos[1] <= (pistapos[1] + dp.Getlongbiascd() / 2))
                            {
                                inTasc = true;
                                this.alt = z;
                            }
                            else
                            {
                                if (pos[1] <= ymax && pos[1] >= (pistapos[1] +
dp.Getlongbiascd() / 2))
                                {
                                    double y0 = Math.Abs(pos[0] - xx1) *
(dp.Getdivascd() / 100) + (pistapos[1] + dp.Getlongbiascd() / 2);
                                    if (pos[1] <= y0)
                                    {
                                        inTasc = true;
                                        this.alt = z;
                                    }
                                }
                                else
                                {
                                    double y0 = Math.Abs(pos[0] - xx1) * (-
dp.Getdivascd() / 100) + (pistapos[1] - dp.Getlongbiascd() / 2);
```

```
                                    if (pos[1] >= y0)
                                    {
                                        inTasc = true;
                                        this.alt = z;
                                    }
                                }
                            }
                        }
                    }
                    if (inTasc == false)
                    {
                        if ((pos[0] >= xmin && pos[0] <= xx2) || (pos[0] >= xx2
&& pos[0] <= xmax))
                        {
                            double z = Math.Abs(pos[0] - xx1) * dp.Getdivascd() /
100 + pistapos[2];
                            if (pos[2] >= z)
                            {
                                inTasc = true;
                                this.alt = z;
                            }
                        }
                    }
                }
                else
                {
                    double z = Math.Abs(pos[0] - xx1) * dp.Getdivascd() / 100 +
pistapos[2];
                    if (pos[2] >= z)
                    {
                        if (pos[1] >= (pistapos[1] - dp.Getlongbiascd() / 2) &&
pos[1] <= (pistapos[1] + dp.Getlongbiascd() / 2))
                        {
                            inTasc = true;
                            this.alt = z;
                        }
                        else
                        {
                            if (pos[1] <= ymax && pos[1] >= (pistapos[1] +
dp.Getlongbiascd() / 2))
                            {
                                double y0 = Math.Abs(pos[0] - xx1) *
(dp.Getdivascd() / 100) + (pistapos[1] + dp.Getlongbiascd() / 2);
                                if (pos[1] <= y0)
                                {
                                    inTasc = true;
                                    this.alt = z;
                                }
                            }
                            else
                            {
                                double y0 = Math.Abs(pos[0] - xx1) * (-
dp.Getdivascd() / 100) + (pistapos[1] - dp.Getlongbiascd() / 2);
                                if (pos[1] >= y0)
                                {
                                    inTasc = true;
                                    this.alt = z;
                                }
                            }
                        }
                    }
                }
```

```
            }
            return inTasc;
        }


        //INTERSECCIONES RADIOELÉCTRICAS
        public bool radioint(SRad srad, double [] pos , double [] radpos)
        {
            bool RadioInt = false;
            xmin = radpos[0] - srad.Getzseg();
            xmax = radpos[0] + srad.Getzseg();
            ymin = radpos[1] - srad.Getzseg();
            ymax = radpos[1] + srad.Getzseg();
            if ((pos[0] >= xmin && pos[0] <= xmax) && (pos[1] >= ymin && pos[1]
<= ymax) && pos[2]>=radpos[2])
            {
                if (pos[0] >= xmin && pos[0] <= (radpos[0]))
                {
                    double x1 = Math.Sqrt(Math.Pow(srad.Getzseg(), 2) -
Math.Pow((pos[1] - (radpos[1])), 2)) + radpos[0];
                    double x2 = -Math.Sqrt(Math.Pow(srad.Getzseg(), 2) -
Math.Pow((pos[1] - (radpos[1])), 2)) + radpos[0];
                    if (pos[0] < 0)
                    {
                        if (pos[0] >= x2)
                        {
                            RadioInt = true;
                        }
                    }
                    else
                    {
                        if (pos[0] >= x1)
                        {
                           RadioInt = true;
                        }
                    }
                }
                if (pos[0] >= (radpos[0]) && pos[0] <= xmax)
                {
                    double x1 = Math.Sqrt(Math.Pow(srad.Getzseg(), 2) -
Math.Pow((pos[1] - (radpos[1])), 2)) + radpos[0];
                    double x2 = -Math.Sqrt(Math.Pow(srad.Getzseg(), 2) -
Math.Pow((pos[1] - (radpos[1])), 2)) + radpos[0];
                    if (pos[0] < 0)
                    {
                        if (pos[0] <= x2)
                        {
                            RadioInt = true;
                        }
                    }
                    else
                    {
                        if (pos[0] <= x1)
                        {
                            RadioInt = true;
                        }
                    }
                }
                this.alt = radpos[2];
            }
            if (RadioInt == false)
            {
```

```
                //Variable que indicará el valor del radio necesario para que la
superficie cubra la altura requerida de la superficie de limitación de alturas
                xmin = radpos[0] - (srad.Getzalt());
                xmax = radpos[0] + (srad.Getzalt());
                ymin = radpos[1] - (srad.Getzalt());
                ymax = radpos[1] + (srad.Getzalt());
                if ((pos[0] >= xmin && pos[0] <= xmax) && (pos[1] >= ymin &&
pos[1] <= ymax))
                {
                    double radiopen;
                    if (pos[0] >= (radpos[0]) && pos[0] <= xmax)
                    {
                        radiopen = -srad.Getzseg() + Math.Sqrt(Math.Pow((pos[0] -
(radpos[0])), 2) + Math.Pow((pos[1] - radpos[1]), 2));
                        if ((radiopen + srad.Getzseg()) <= srad.Getzalt())
                        {
                            double z = (srad.Getpen() / 100) * radiopen + 45;
                            if (pos[2] >= z)
                            {
                                if (pos[1] <= (radpos[1] - srad.Getzseg()) ||
pos[1] >= (radpos[1] + srad.Getzseg()))
                                {
                                    RadioInt = true;
                                    this.alt = z;
                                }
                                else
                                {
                                    double x1 =
Math.Sqrt(Math.Pow(srad.Getzseg(), 2) - Math.Pow((pos[1] - (radpos[1])), 2)) +
radpos[0];
                                    double x2 = -
Math.Sqrt(Math.Pow(srad.Getzseg(), 2) - Math.Pow((pos[1] - (radpos[1])), 2)) +
radpos[0];
                                    if (pos[0] < 0)
                                    {
                                        if (pos[0] >= x2)
                                        {
                                            RadioInt = true;
                                            this.alt = z;
                                        }
                                    }
                                    else
                                    {
                                        if (pos[0] >= x1)
                                        {
                                            RadioInt = true;
                                            this.alt = z;
                                        }
                                    }
                                }
                            }
                        }
                    }
                    else
                    {
                        radiopen = -srad.Getzseg() + Math.Sqrt(Math.Pow((pos[0] -
(radpos[0])), 2) + Math.Pow((pos[1] - radpos[1]), 2));
                        if ((radiopen + srad.Getzseg()) <= srad.Getzalt())
                        {
                            double z = (srad.Getpen() / 100) * radiopen + 45;
                            if (pos[2] >= z)
                            {
```

```
                                if (pos[1] <= (radpos[1] - srad.Getzseg()) ||
pos[1] >= (radpos[1] + srad.Getzseg()))
                                {
                                    RadioInt = true;
                                    this.alt = z;
                                }
                                else
                                {
                                    double x1 =
Math.Sqrt(Math.Pow(srad.Getzseg(), 2) - Math.Pow((pos[1] - (radpos[1])), 2)) +
radpos[0];
                                    double x2 = -
Math.Sqrt(Math.Pow(srad.Getzseg(), 2) - Math.Pow((pos[1] - (radpos[1])), 2)) +
radpos[0];
                                    if (pos[0] < 0)
                                    {
                                        if (pos[0] <= x2)
                                        {
                                            RadioInt = true;
                                            this.alt = z;
                                        }
                                    }
                                    else
                                    {
                                        if (pos[0] <= x1)
                                        {
                                            RadioInt = true;
                                            this.alt = z;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            return RadioInt;
        }

        //ILS/LOC
        public bool ilocint(double[] pos, double[] ipos, double posumbral, double
sentido, double longitudpista, double[] pistapos,double ang)
        {
            bool IlocInt = false;
            double d;
            double[] tempipos = { ipos[0], ipos[1],ipos[2] };
            ipos = giro(tempipos, ang);
            if (sentido == -1)
            {
                x1 = pistapos[0] - longitudpista / 2 + posumbral;
                x2 = ipos[0] + (-1) * Math.Sqrt(Math.Pow((-x1 + ipos[0]), 2));
                d = Math.Abs(x1 - ipos[0]);
            }
            else
            {
                x2 = pistapos[0] + longitudpista / 2 - posumbral;
                x1 = ipos[0] + Math.Abs(ipos[0] - x2);
                d = Math.Abs(ipos[0] - x2);
            }
            xmin = ipos[0] - 2500;
            xmax = ipos[0] + 2500;
            y1 = ipos[1] - Math.Abs(d) * Math.Tan(30 * 2 * Math.PI / 360);
```

```
            y2 = ipos[1] + Math.Abs(d) * Math.Tan(30 * 2 * Math.PI / 360);
            ymin = ipos[1] - 2500 * Math.Tan(30 * 2 * Math.PI / 360);
            ymax = ipos[1] + 2500 * Math.Tan(30 * 2 * Math.PI / 360);
            yar = ipos[1] + 500;
            yab = ipos[1] - 500;
            x11 = ipos[0] + 500 / Math.Tan(30 * 2 * Math.PI / 360);
            x22 = ipos[0] - 500 / Math.Tan(30 * 2 * Math.PI / 360);
            if ((pos[0] >= xmin && pos[0] <= xmax) && (pos[1] >= ymin && pos[1]
<= ymax))
            {
                if ((pos[0] >= x1 && pos[0] <= x2) && (pos[1] >= y1 && pos[1] <=
y2) && pos[2] >= ipos[2])
                {
                    IlocInt = true;
                    this.alt = ipos[2];
                }
                else
                {
                    if ((pos[1] >= y2 && pos[1] <= yar) && IlocInt == false)
                    {
                        if (pos[0] >= x1 && pos[0] <= x2)
                        {
                            double z = Math.Abs(pos[1] - y2) * 2 / 100 + ipos[2];
                            if (pos[2] >= z)
                            {
                                IlocInt = true;
                                this.alt = z;
                            }
                        }
                        else
                        {
                            if (pos[0] >= x1 && pos[0] <= (ipos[0] + 500 /
(Math.Tan(30 * 2 * Math.PI / 360))) && IlocInt == false)
                            {
                                double y0 = y2 + (Math.Tan(30 * 2 * Math.PI /
360) * Math.Abs(pos[0] - x1));
                                if (pos[1] >= y0)
                                {
                                    double z = Math.Abs(pos[1] - y2) * 2 / 100 +
ipos[2];
                                    if (pos[2] >= z)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                }
                            }
                            if ((pos[0] <= x2 && pos[0] >= (ipos[0] - 500)) &&
IlocInt == false)
                            {
                                double y0 = y2 + (Math.Tan(30 * 2 * Math.PI /
360) * Math.Abs(pos[0] - x2));
                                if (pos[1] >= y0)
                                {
                                    double z = Math.Abs(pos[1] - y2) * 2 / 100 +
ipos[2];
                                    if (pos[2] >= z)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                }
```

```
                        }
                    }
                }
                else
                {
                    if ((pos[1] >= yab && pos[1] <= y1) && IlocInt == false)
                    {
                        if (pos[0] >= x1 && pos[0] <= x2)
                        {
                            double z = Math.Abs(pos[1] - y1) * 2 / 100 +
ipos[2];
                            if (pos[2] >= z)
                            {
                                IlocInt = true;
                                this.alt = z;
                            }
                        }
                        else
                        {
                            if (pos[0] >= x1 && pos[0] <= (ipos[0] + 500 /
(Math.Tan(30 * 2 * Math.PI / 360))) && IlocInt == false)
                            {
                                double y0 = y1 - (Math.Tan(30 * 2 * Math.PI /
360) * Math.Abs(pos[0] - x1));
                                if (pos[1] <= y0)
                                {
                                    double z = Math.Abs(pos[1] - y1) * 2 /
100 + ipos[2];
                                    if (pos[2] >= z)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                }
                            }
                            if ((pos[0] <= x2 && pos[0] >= (ipos[0] - 500))
&& IlocInt == false)
                            {
                                double y0 = y1 - (Math.Tan(30 * 2 * Math.PI /
360) * Math.Abs(pos[0] - x2));
                                if (pos[1] <= y0)
                                {
                                    double z = Math.Abs(pos[1] - y1) * 2 /
100 + ipos[2];
                                    if (pos[2] >= z)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                }
                            }
                        }
                    }
                }
                if ((pos[0] >= x1 && pos[0] <= xmax) && IlocInt == false)
                {
                    double z = Math.Abs(pos[0] - x1) * 2 / 100 + ipos[2];
                    if (pos[2] >= z)
                    {
                        if (pos[1] >= y1 && pos[2] <= y2)
                        {
                            IlocInt = true;
```

```
                                this.alt = z;
                            }
                            else
                            {
                                if (pos[1] >= y2 && pos[1] <= ymax)
                                {
                                    double y0 = y2 + (Math.Tan(30 * 2 * Math.PI /
360) * Math.Abs(pos[0] - x1));
                                    if (pos[1] <= y0)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                    else
                                    {
                                        if (pos[1] >= ymin && pos[1] <= y1)
                                        {
                                            y0 = y1 - (Math.Tan(30 * 2 * Math.PI
/ 360) * Math.Abs(pos[0] - x1));
                                            if (pos[1] >= y0)
                                            {
                                                IlocInt = true;
                                                this.alt = z;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                        else
                        {
                            if (pos[0] >= xmin && pos[0] <= x2)
                            {
                                double z = Math.Abs(pos[0] - x2) * 2 / 100 + ipos[2];
                                if (pos[2] >= z)
                                {
                                    if (pos[1] >= y1 && pos[2] <= y2)
                                    {
                                        IlocInt = true;
                                        this.alt = z;
                                    }
                                    else
                                    {
                                        if (pos[1] >= y2 && pos[1] <= ymax)
                                        {
                                            double y0 = y2 + (Math.Tan(30 * 2 *
Math.PI / 360) * Math.Abs(pos[0] - x2));
                                            if (pos[1] <= y0)
                                            {
                                                IlocInt = true;
                                                this.alt = z;
                                            }
                                            else
                                            {
                                                if (pos[1] >= ymin && pos[1] <= y1)
                                                {
                                                    y0 = y1 - (Math.Tan(30 * 2 *
Math.PI / 360) * Math.Abs(pos[0] - x2));
                                                    if (pos[1] >= y0)
                                                    {
                                                        IlocInt = true;
```

```
                                                this.alt = z;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            return IlocInt;
        }

        //ILS/GP
        public bool igpint(double[] pos, double[]ipos,double posumbral, double
longitudpista, double[]pistapos, double ancho, double ang)
        {
            bool IgpInt = false;
            double[] tempipos = { ipos[0], ipos[1], ipos[2] };
            ipos = giro(tempipos, ang);
            y1 = ipos[1] + (Math.Abs(ipos[1] - pistapos[1]) + ancho / 2 + 200);
            y2 = ipos[1] - (Math.Abs(ipos[1] - pistapos[1]) + ancho / 2 + 200);
            if (ipos[0] < pistapos[0])
            {
                x1 = ipos[0] - Math.Abs(pistapos[0] - longitudpista / 2 +
posumbral) - 600;
                x2 = ipos[0] + Math.Abs(pistapos[0] - longitudpista / 2 +
posumbral) + 200;
                xmax = x2;
                xmin = ipos[0] - 5000 - Math.Abs(ipos[1] - pistapos[1]);
                ymax = ipos[1] + Math.Abs(xmin - ipos[0]) * Math.Tan(20 * 2 *
Math.PI / 360);
                ymin  = ipos[1] - Math.Abs(xmin - ipos[0]) * Math.Tan(20 * 2 *
Math.PI / 360);
                //Buscamos los puntos donde intersectan las zonas de seguridad y
de limitación de alturas
                m1 = (ymax - ipos[1]) / (xmin - ipos[0]);
                m2 = (ymin - ipos[1]) / (xmin - ipos[0]);
                b1 = ymax - m1 * xmin;
                b2 = ymin - m2 * xmin;
                //Miramos el valor de la x para el valor de la y a partir de la
ecuación de su recta a partir de la pendiente y la cte
                xx1 = (y1 - b1) / m1;
                if (xx1 >= x1)
                {
                    yy1 = y1;
                }
                else
                {
                    xx1 = x1;
                    yy1 = xx1 * m1 + b1;
                }
                xx2 = (y2 - b2) / m2;
                if (xx2 >= x1)
                {
                    yy2 = y2;
                }
                else
                {
                    xx2 = x1;
                    yy2 = xx2 * m2 + b2;
```

```
                }
                if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= xmin &&
pos[0] <= ipos[0]))
                {
                    if (pos[1] >= yy1)
                    {
                        double y0 = yy1 + (Math.Tan(20 * 2 * Math.PI / 360) *
Math.Abs(pos[0] - xx1));
                        if (pos[1] <= y0)
                        {
                            double z;
                            if (pos[0] > x1)
                            {
                                z = ipos[2];
                            }
                            else
                            {
                                z = ipos[2] + Math.Abs(pos[0] - x1) * 2 / 100;
                            }
                            if (pos[2] >= z)
                            {
                                IgpInt = true;
                                this.alt = z;
                            }
                        }
                    }
                    if (pos[1] <= yy2)
                    {
                        double y0 = yy2 - (Math.Tan(20 * 2 * Math.PI / 360) *
Math.Abs(pos[0] - xx2));
                        if (pos[1] >= y0)
                        {
                            double z;
                            if (pos[0] > x1)
                            {
                                z = ipos[2];
                            }
                            else
                            {
                                z = ipos[2] + Math.Abs(pos[0] - x1) * 2 / 100;
                            }
                            if (pos[2] >= z)
                            {
                                IgpInt = true;
                                this.alt = z;
                            }
                        }
                    }
                }
                yar = m1 * x1 + b1;
                yab = m2 * x1 + b2;
            }
            else
            {
                x2 = ipos[0] + Math.Abs(pistapos[0] + longitudpista / 2 -
posumbral) + 600;
                x1 = ipos[0] - Math.Abs(pistapos[0] + longitudpista / 2 -
posumbral) - 200;
                xmin = x1;
                xmax = ipos[0] + 5000 + Math.Abs(ipos[1] - pistapos[1]);
                ymax = ipos[1] + Math.Abs(xmax - ipos[0]) * Math.Tan(20 * 2 *
Math.PI / 360);
```

```
                ymin = ipos[1] - Math.Abs(xmax - ipos[0]) * Math.Tan(20 * 2 *
Math.PI / 360);
                //Buscamos los puntos donde intersectan las zonas de seguridad y
de limitación de alturas
                m1 = (ymax - ipos[1]) / (xmax - ipos[0]);
                m2 = (ymin - ipos[1]) / (xmax - ipos[0]);
                b1 = ymax - m1 * xmax;
                b2 = ymin - m2 * xmax;
                //Miramos el valor de la x para el valor de la y a partir de la
ecuación de su recta a partir de la pendiente y la cte
                xx1 = (y1 - b1) / m1;
                if (xx1 <= x2)
                {
                    yy1 = y1;
                }
                else
                {
                    xx1 = x2;
                    yy1 = xx1 * m1 + b1;
                }
                xx2 = (y2 - b2) / m2;
                if (xx2 <= x2)
                {
                    yy2 = y2;
                }
                else
                {
                    xx2 = x2;
                    yy2 = xx2 * m2 + b2;
                }
                if ((pos[1] >= ymin && pos[1] <= ymax) && (pos[0] >= ipos[0] &&
pos[0] <= xmax))
                {
                    if (pos[1] >= yy1)
                    {
                        double y0 = yy1 + (Math.Tan(20 * 2 * Math.PI / 360) *
Math.Abs(pos[0] - xx1));
                        if (pos[1] <= y0)
                        {
                            double z;
                            if (pos[0] < x2)
                            {
                                z = ipos[2];
                            }
                            else
                            {
                                z = ipos[2] + Math.Abs(pos[0] - x2) * 2 / 100;
                            }
                            if (pos[2] >= z)
                            {
                                IgpInt = true;
                                this.alt = z;
                            }
                        }
                    }
                    if (pos[1] <= yy2)
                    {
                        double y0 = yy2 - (Math.Tan(20 * 2 * Math.PI / 360) *
Math.Abs(pos[0] - xx2));
                        if (pos[1] >= y0)
                        {
                            double z;
```

```
                            if (pos[0] < x2)
                            {
                                z = ipos[2];
                            }
                            else
                            {
                                z = ipos[2] + Math.Abs(pos[0] - x2) * 2 / 100;
                            }
                            if (pos[2] >= z)
                            {
                                IgpInt = true;
                                this.alt = z;
                            }
                        }
                    }
                }
                yar = m1 * x2 + b1;
                yab = m2 * x2 + b2;
            }

            if ((pos[0] >= x1 && pos[0] <= x2) && (pos[1] >= y1 && pos[1] <= y2)
&& pos[2] >= ipos[2] && IgpInt == false)
            {
                IgpInt = true;
                this.alt = ipos[2];
            }
            return IgpInt;
        }
    }
}
```

IntersecInfo.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LibreriaDimPen;

namespace LibreriaIntersecciones
{
    public partial class IntersecInfo : Form
    {
        //Long.pista
        string lpista;
        //Clave pista
        int clpista;
        //Categoría pista
        string catepista;
        //Tipo de aproximación
        object apx;
        //Vector de booleanos que regularán la visibilidad de ciertos botones del
FORM
        bool[] visible = { false, false };
        //Variable donde se encuentran las dimesiones de las servidumbres físicas
        DimPen dp;
        //MAtriz donde están las coordenadas de los obstáculos
        double[,] pos = null;
        //Vector donde están las coordenadas de la pista
        double[] pistapos;
        //Matriz donde se hayan las variables booleanas que indicarán si existe
intersección entre los obstáculos y las SLO
        bool[,] intersecciones;
        //Valor de aproximación
        string aproxval;
        //SEntidos de aterrizaje y despegue
        double sentaterrizaje;
        double sentdespegue;
        //Clase a partir de la cual llamaremos los métodos para saber si hay
intersecciones
        Intersecciones its = new Intersecciones();
        //Ángulo de translación
        double ang;
        //Posición del umbral
        double posumbral;
        //Características dimensionales de la franja
        Franjadim fjd;
        //MAtriz donde se hayan las alturas donde se producen las intersecciones
        double[,] alt = null;
        //Variable que indicará la posición del obstáculo, los datos del cual se
pueden apreciar en el form, este variable se utilizará cuando se tenga más de un
obstáculo
        int posvector;
        //Clase donde estará guardado la identificación y las coordenadas de los
obstáculos
        OBS[] Oblist;
        public IntersecInfo()
        {
            InitializeComponent();
```

```
            siguientebuttom.Visible = visible[1];
            anteriorbuttom.Visible = visible[1];
        }

        private void Cerrar_Click(object sender, EventArgs e)
        {
            Close();
        }

        //MÉTODOS SET
        public void Setinfo(string lpista, int clpista, string catepista, object
apx, DimPen dp, double[,] pistapos, string aproxval, double sentaterrizaje,
double sentdespegue, double umbral, Franjadim fjd, double ang,bool obst, OBS[]
Obslist)
        {
            this.lpista = lpista;
            this.clpista = clpista;
            this.catepista = catepista;
            this.apx = apx;
            this.dp = dp;
            this.pistapos = new double[3];
            this.pistapos[0] = pistapos[0, 0];
            this.pistapos[1] = pistapos[0, 1];
            this.pistapos[2] = pistapos[0, 2];
            this.aproxval = aproxval;
            this.sentaterrizaje = sentaterrizaje;
            this.sentdespegue = sentdespegue;
            this.posumbral = umbral;
            this.fjd = fjd;
            this.ang = ang;
            this.visible[0] = obst;
            if (this.visible[0] == true)
            {
                this.Oblist = new OBS[Obslist.Length];
                this.Oblist = Obslist;
                if (this.Oblist.Length > 1)
                {
                    this.visible[1] = true;
                }
            }
            siguientebuttom.Visible = visible[1];
            anteriorbuttom.Visible = visible[1];
            longitudpista.Text = "Longitud:" + this.lpista;
            clave.Text = "Clave:" + Convert.ToString(this.clpista);
            categoria.Text = "Categoría:" + this.catepista;
            aprox.Text = "Tipo de aproximación:" + this.apx;
            Xpista.Text = "X:" + Convert.ToString(this.pistapos[0]);
            Ypista.Text = "Y:" + Convert.ToString(this.pistapos[1]);
            Zpista.Text = "Z:" + Convert.ToString(this.pistapos[2]);
        }

        private void borrarObstáculosToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            bool[] visible = { false, false };
            siguientebuttom.Visible = visible[1];
            anteriorbuttom.Visible = visible[1];
        }

        private void modificarconsultarObstáculosToolStripMenuItem_Click(object
sender, EventArgs e)
        {
```

```
            if (visible[0] == true)
            {
                Obstaculos vnt2 = new Obstaculos();
                vnt2.Setinfo(this.Oblist);
                vnt2.rellenar(this.Oblist);
                vnt2.ShowDialog();
                visible[0] = vnt2.Getrellenado();
                this.Oblist = vnt2.Getobs();
                if (visible[0] == true)
                {
                    //Variable temporal
                    double[,] temppos = new double[this.Oblist.Length,3];
                    for (int i = 0; i < this.Oblist.Length; i++)
                    {
                        temppos[i, 0] = this.Oblist[i].Getpos()[0];
                        temppos[i, 1] = this.Oblist[i].Getpos()[1];
                        temppos[i, 2] = this.Oblist[i].Getpos()[2];
                    }
                    if (temppos != null)
                    {
                        this.pos = temppos;
                    }
                    if (this.pos.Length / 3 > 1)
                    {
                        visible[1] = true;
                    }
                    else
                    {
                        visible[1] = false;
                    }
                    siguientebuttom.Visible = visible[1];
                    anteriorbuttom.Visible = visible[1];
                }
                vaciarpanel();
            }
            else
            {
                MessageBox.Show("Por favor, cargue o introduzca antes las
coordenadas de los obstáculos.");
            }
        }

        public double[,] Getpos()
        {
            return this.pos;
        }

        private void Intersecciones_Click(object sender, EventArgs e)
        {
            Interseccioness();
        }
        public void Interseccioness()
        {
            if (visible[0] == true)
            {
                //Variable temporal con la posición de la pista transaladadas.
                double[] temppistapos = new double[3];
                temppistapos[0] = this.pistapos[0];
                temppistapos[1] = this.pistapos[1];
                temppistapos[2] = this.pistapos[2];
                temppistapos = its.giro(temppistapos, this.ang);
                if (visible[1] == false)
```

```
                {
                    //Variable temporal con las posiciones del obstáculo
                    double[] tempposi = new double[3];
                    tempposi[0] = this.pos[0, 0];
                    tempposi[1] = this.pos[0, 1];
                    tempposi[2] = this.pos[0, 2];
                    tempposi = its.giro(tempposi, this.ang);
                    if (this.aproxval == "Visual" || this.aproxval == "NoPrec")
                    {
                        this.intersecciones = new bool[1, 6];
                        this.alt = new double[1, 6];
                        this.intersecciones[0, 0] =
this.its.intconi(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos, dp.Getpencon(), dp.Getaltcon());
                        this.alt[0, 0] = this.its.Getalt();
                        this.intersecciones[0, 1] =
this.its.inthoint(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos);
                        this.alt[0, 1] = this.its.Getalt();
                        this.intersecciones[0, 2] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[0];
                        this.alt[0, 2] = this.its.Getalt();
                        this.intersecciones[0, 3] = this.its.inttrans(dp, fjd,
temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[0];
                        this.alt[0, 3] = this.its.Getalt();
                        this.intersecciones[0, 4] = this.its.intsai(dp, tempposi,
temppistapos, Convert.ToDouble(lpista), this.sentaterrizaje, this.posumbral,
this.aproxval);
                        this.alt[0, 4] = this.its.Getalt();
                        this.intersecciones[0, 5] = this.its.intasc(dp, tempposi,
temppistapos, Convert.ToDouble(lpista), this.sentdespegue);
                        this.alt[0, 5] = this.its.Getalt();
                        if (this.intersecciones[0, 0] == false)
                        {
                            coninfo.Text = "El obstáculo NO intersecta con la
cónica.";
                            conalt.Text = "";
                        }
                        else
                        {
                            coninfo.Text = "El obstáculo SI intersecta con la
cónica.";
                            conalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,0]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 1] == false)
                        {
                            Hintinfo.Text = "El obstáculo NO intersecta con la
horizontal interna.";
                            hintalt.Text = "";
                        }
                        else
                        {
                            Hintinfo.Text = "El obstáculo SI intersecta con la
horizontal interna.";
                            hintalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,1]),2))
+ "m.";
                        }
```

```
                        if (this.intersecciones[0, 2] == false)
                        {
                            aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                            apalt.Text = "";
                        }
                        else
                        {
                            aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                            apalt.Text ="La intersección se produce a una altura
de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,2]),2)) +
"m.";
                        }
                        if (this.intersecciones[0, 3] == false)
                        {
                            transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                            transalt.Text = "";
                        }
                        else
                        {
                            transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                            transalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,3]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 4] == false)
                        {
                            SAIinfo.Text = "El obstáculo NO intersecta con la
SAI.";
                            SAIalt.Text = "";
                        }
                        else
                        {
                            SAIinfo.Text = "El obstáculo SI intersecta con la
SAI.";
                            SAIalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,4]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 5] == false)
                        {
                            acsdinfo.Text = "El obstáculo NO intersecta con la
superficie ACSD.";
                            ascdalt.Text = "";
                        }
                        else
                        {
                            acsdinfo.Text = "El obstáculo SI intersecta con la
superficie ASCD.";
                            ascdalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,5]),2))
+ "m.";
                        }
                        aintinfo.Text = "--";
                        transintinfo.Text = "--";
                        X.Text = "X:" + Convert.ToString(this.pos[0, 0]);
                        Y.Text = "Y:" + Convert.ToString(this.pos[0, 1]);
                        Z.Text = "Z:" + Convert.ToString(this.pos[0, 2]);
                    }
```

```
                    else
                    {
                        this.intersecciones = new bool[1, 8];
                        this.alt = new double[1, 8];
                        this.intersecciones[0, 0] =
this.its.intconi(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos, dp.Getpencon(), dp.Getaltcon());
                        this.alt[0, 0] = this.its.Getalt();
                        this.intersecciones[0, 1] =
this.its.inthoint(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos);
                        this.alt[0, 1] = this.its.Getalt();
                        this.intersecciones[0, 2] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[0];
                        this.alt[0, 2] = this.its.Getalt();
                        this.intersecciones[0, 3] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[1];
                        this.alt[0, 3] = this.its.Getalt();
                        this.intersecciones[0, 4] = this.its.inttrans(dp, fjd,
temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[0];
                        this.alt[0, 4] = this.its.Getalt();
                        this.intersecciones[0, 5] = this.its.inttrans(dp, fjd,
temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[1];
                        this.alt[0, 5] = this.its.Getalt();
                        this.intersecciones[0, 6] = this.its.intsai(dp, tempposi,
temppistapos, Convert.ToDouble(lpista), this.sentaterrizaje, this.posumbral,
this.aproxval);
                        this.alt[0, 6] = this.its.Getalt();
                        this.intersecciones[0, 7] = this.its.intasc(dp, tempposi,
temppistapos, Convert.ToDouble(lpista), this.sentdespegue);
                        this.alt[0, 7] = this.its.Getalt();
                        if (this.intersecciones[0, 0] == false)
                        {
                            coninfo.Text = "El obstáculo NO intersecta con la
cónica.";
                            conalt.Text = "";
                        }
                        else
                        {
                            coninfo.Text = "El obstáculo SI intersecta con la
cónica.";
                            conalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,0]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 1] == false)
                        {
                            Hintinfo.Text = "El obstáculo NO intersecta con la
horizontal interna.";
                            hintalt.Text = "";
                        }
                        else
                        {
                            Hintinfo.Text = "El obstáculo SI intersecta con la
horizontal interna.";
                            hintalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,1]),2))
+ "m.";
```

```
                        }
                        if (this.intersecciones[0, 2] == false)
                        {
                            aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                            apalt.Text = "";
                        }
                        else
                        {
                            aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                            apalt.Text = "La intersección se produce a una altura
de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,2]),2)) +
"m.";
                        }
                        if (this.intersecciones[0, 3] == false)
                        {
                            aintinfo.Text = "El obstáculo NO intersecta con la
superficie de Ap.interna.";
                            apialt.Text = "";
                        }
                        else
                        {
                            aintinfo.Text = "El obstáculo SI intersecta con la
superficie de Ap.interna.";
                            apialt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,3]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 4] == false)
                        {
                            transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                            transalt.Text = "";
                        }
                        else
                        {
                            transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                            transalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,4]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 5] == false)
                        {
                            transintinfo.Text = "El obstáculo NO intersecta con
la superficie de T.interna.";
                            transintalt.Text = "";
                        }
                        else
                        {
                            transintinfo.Text = "El obstáculo SI intersecta con
la superficie de T.interna.";
                            transintalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,5]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 6] == false)
                        {
                            SAIinfo.Text = "El obstáculo NO intersecta con la
SAI.";
                            SAIalt.Text = "";
```

```
                        }
                        else
                        {
                            SAIinfo.Text = "El obstáculo SI intersecta con la
SAI.";
                            SAIalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,6]),2))
+ "m.";
                        }
                        if (this.intersecciones[0, 7] == false)
                        {
                            acsdinfo.Text = "El obstáculo NO intersecta con la
superficie ACSD.";
                            ascdalt.Text = "";
                        }
                        else
                        {
                            acsdinfo.Text = "El obstáculo SI intersecta con la
superficie ASCD.";
                            ascdalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0,7]),2))
+ "m.";
                        }
                        X.Text = "X:" + Convert.ToString(this.pos[0, 0]);
                        Y.Text = "Y:" + Convert.ToString(this.pos[0, 1]);
                        Z.Text = "Z:" + Convert.ToString(this.pos[0, 2]);
                    }
                }
                else
                {
                    if (this.aproxval == "Visual" || this.aproxval == "NoPrec")
                    {
                        this.intersecciones = new bool[this.pos.Length / 3, 6];
                        this.alt = new double[this.pos.Length / 3, 6];
                        double[] tempposi = new double[3];
                        for (int i = 0; i < this.pos.Length / 3; i++)
                        {
                            tempposi[0] = this.pos[i, 0];
                            tempposi[1] = this.pos[i, 1];
                            tempposi[2] = this.pos[i, 2];
                            tempposi = its.giro(tempposi, this.ang);
                            this.intersecciones[i, 0] =
this.its.intconi(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos, dp.Getpencon(), dp.Getaltcon());
                            this.alt[i, 0] = this.its.Getalt();
                            this.intersecciones[i, 1] =
this.its.inthoint(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos);
                            this.alt[i, 1] = this.its.Getalt();
                            this.intersecciones[i, 2] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[0];
                            this.alt[i, 2] = this.its.Getalt();
                            this.intersecciones[i, 3] = this.its.inttrans(dp,
fjd, temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[0];
                            this.alt[i, 3] = this.its.Getalt();
                            this.intersecciones[i, 4] = this.its.intsai(dp,
tempposi, temppistapos, Convert.ToDouble(lpista), this.sentaterrizaje,
this.posumbral, this.aproxval);
                            this.alt[i, 4] = this.its.Getalt();
```

```
                            this.intersecciones[i, 5] = this.its.intasc(dp,
tempposi, temppistapos, Convert.ToDouble(lpista), this.sentdespegue);
                            this.alt[i, 5] = this.its.Getalt();
                        }
                        if (this.intersecciones[0, 0] == false)
                        {
                            coninfo.Text = "El obstáculo NO intersecta con la
cónica.";
                            conalt.Text = "";
                        }
                        else
                        {
                            coninfo.Text = "El obstáculo SI intersecta con la
cónica.";
                            conalt.Text = "La intersección se produce a una
altura de:" + Convert.ToString(this.alt[0, 0]) + "m.";
                        }
                        if (this.intersecciones[0, 1] == false)
                        {
                            Hintinfo.Text = "El obstáculo NO intersecta con la
horizontal interna.";
                            hintalt.Text = "";
                        }
                        else
                        {
                            Hintinfo.Text = "El obstáculo SI intersecta con la
horizontal interna.";
                            hintalt.Text = "La intersección se produce a una
altura de:" + Convert.ToString(this.alt[0, 1]) + "m.";
                        }
                        if (this.intersecciones[0, 2] == false)
                        {
                            aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                            apalt.Text = "";
                        }
                        else
                        {
                            aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                            apalt.Text = "La intersección se produce a una altura
de:" + Convert.ToString(this.alt[0, 2]) + "m.";
                        }
                        if (this.intersecciones[0, 3] == false)
                        {
                            transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                            transalt.Text = "";
                        }
                        else
                        {
                            transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                            transalt.Text = "La intersección se produce a una
altura de:" + Convert.ToString(this.alt[0, 3]) + "m.";
                        }
                        if (this.intersecciones[0, 4] == false)
                        {
                            SAIinfo.Text = "El obstáculo NO intersecta con la
SAI.";
                            SAIalt.Text = "";
                        }
```

```
                        else
                        {
                            SAIinfo.Text = "El obstáculo SI intersecta con la
SAI.";
                            SAIalt.Text = "La intersección se produce a una
altura de:" + Convert.ToString(this.alt[0, 4]) + "m.";
                        }
                        if (this.intersecciones[0, 5] == false)
                        {
                            acsdinfo.Text = "El obstáculo NO intersecta con la
superficie ACSD.";
                            ascdalt.Text = "";
                        }
                        else
                        {
                            acsdinfo.Text = "El obstáculo SI intersecta con la
superficie ASCD.";
                            ascdalt.Text = "La intersección se produce a una
altura de:" + Convert.ToString(this.alt[0, 5]) + "m.";
                        }
                        aintinfo.Text = "--";
                        transintinfo.Text = "--";
                        X.Text = "X:" + Convert.ToString(this.pos[0, 0]);
                        Y.Text = "Y:" + Convert.ToString(this.pos[0, 1]);
                        Z.Text = "Z:" + Convert.ToString(this.pos[0, 2]);
                        this.posvector = 0;
                    }
                    else
                    {
                        this.intersecciones = new bool[this.pos.Length / 3, 8];
                        this.alt = new double[this.pos.Length / 3, 8];
                        double[] tempposi = new double[3];
                        for (int i = 0; i < this.pos.Length / 3; i++)
                        {
                            tempposi[0] = this.pos[i, 0];
                            tempposi[1] = this.pos[i, 1];
                            tempposi[2] = this.pos[i, 2];
                            tempposi = its.giro(tempposi, this.ang);
                            this.intersecciones[i, 0] =
this.its.intconi(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos, dp.Getpencon(), dp.Getaltcon());
                            this.alt[i, 0] = this.its.Getalt();
                            this.intersecciones[i, 1] =
this.its.inthoint(Convert.ToDouble(this.lpista), dp.Getradhint(), tempposi,
temppistapos);
                            this.alt[i, 1] = this.its.Getalt();
                            this.intersecciones[i, 2] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[0];
                            this.alt[i, 2] = this.its.Getalt();
                            this.intersecciones[i, 3] =
this.its.intaprox(this.aproxval, dp, tempposi, temppistapos, this.clpista,
Convert.ToDouble(lpista), this.posumbral, this.sentaterrizaje)[1];
                            this.alt[i, 3] = this.its.Getalt();
                            this.intersecciones[i, 4] = this.its.inttrans(dp,
fjd, temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[0];
                            this.alt[i, 4] = this.its.Getalt();
                            this.intersecciones[i, 5] = this.its.inttrans(dp,
fjd, temppistapos, tempposi, this.sentaterrizaje, Convert.ToDouble(lpista),
this.aproxval, this.posumbral)[1];
                            this.alt[i, 5] = this.its.Getalt();
```

```
                            this.intersecciones[i, 6] = this.its.intsai(dp,
tempposi, temppistapos, Convert.ToDouble(lpista), this.sentaterrizaje,
this.posumbral, this.aproxval);
                            this.alt[i, 6] = this.its.Getalt();
                            this.intersecciones[i, 7] = this.its.intasc(dp,
tempposi, temppistapos, Convert.ToDouble(lpista), this.sentdespegue);
                            this.alt[i, 7] = this.its.Getalt();
                        }
                        if (this.intersecciones[0, 0] == false)
                        {
                            coninfo.Text = "El obstáculo NO intersecta con la
cónica.";
                            conalt.Text = "";
                        }
                        else
                        {
                            coninfo.Text = "El obstáculo SI intersecta con la
cónica.";
                            conalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 0]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 1] == false)
                        {
                            Hintinfo.Text = "El obstáculo NO intersecta con la
horizontal interna.";
                            hintalt.Text = "";
                        }
                        else
                        {
                            Hintinfo.Text = "El obstáculo SI intersecta con la
horizontal interna.";
                            hintalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 1]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 2] == false)
                        {
                            aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                            apalt.Text = "";
                        }
                        else
                        {
                            aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                            apalt.Text = "La intersección se produce a una altura
de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 2]), 2)) +
"m.";
                        }
                        if (this.intersecciones[0, 3] == false)
                        {
                            aintinfo.Text = "El obstáculo NO intersecta con la
superficie de Ap.interna.";
                            apialt.Text = "";
                        }
                        else
                        {
                            aintinfo.Text = "El obstáculo SI intersecta con la
superficie de Ap.interna.";
```

```
                            apialt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 3]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 4] == false)
                        {
                            transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                            transalt.Text = "";
                        }
                        else
                        {
                            transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                            transalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 4]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 5] == false)
                        {
                            transintinfo.Text = "El obstáculo NO intersecta con
la superficie de T.interna.";
                            transintalt.Text = "";
                        }
                        else
                        {
                            transintinfo.Text = "El obstáculo SI intersecta con
la superficie de T.interna.";
                            transintalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 5]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 6] == false)
                        {
                            SAIinfo.Text = "El obstáculo NO intersecta con la
SAI.";
                            SAIalt.Text = "";
                        }
                        else
                        {
                            SAIinfo.Text = "El obstáculo SI intersecta con la
SAI.";
                            SAIalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 6]),
2)) + "m.";
                        }
                        if (this.intersecciones[0, 7] == false)
                        {
                            acsdinfo.Text = "El obstáculo NO intersecta con la
superficie ACSD.";
                            ascdalt.Text = "";
                        }
                        else
                        {
                            acsdinfo.Text = "El obstáculo SI intersecta con la
superficie ASCD.";
                            ascdalt.Text = "La intersección se produce a una
altura de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[0, 7]),
2)) + "m.";
                        }
                        X.Text = "X:" + Convert.ToString(this.pos[0, 0]);
                        Y.Text = "Y:" + Convert.ToString(this.pos[0, 1]);
```

```
                        Z.Text = "Z:" + Convert.ToString(this.pos[0, 2]);
                        this.posvector = 0;
                    }
                }
            }
            else
            {
                MessageBox.Show("Por favor introduzca o cargue primero los
obstáculos.");
            }
        }

        private void siguientebuttom_Click(object sender, EventArgs e)
        {
            if (this.posvector == (this.pos.Length / 3 - 1))
            {
                this.posvector = 0;
            }
            else
            {
                this.posvector = this.posvector + 1;
            }
            if (this.aproxval == "Visual" || this.aproxval == "NoPrec")
            {
                if (this.intersecciones[this.posvector, 0] == false)
                {
                    coninfo.Text = "El obstáculo NO intersecta con la cónica.";
                    conalt.Text = "";
                }
                else
                {
                    coninfo.Text = "El obstáculo SI intersecta con la cónica.";
                    conalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 0]) + "m.";
                }
                if (this.intersecciones[this.posvector, 1] == false)
                {
                    Hintinfo.Text = "El obstáculo NO intersecta con la horizontal
interna.";
                    hintalt.Text = "";
                }
                else
                {
                    Hintinfo.Text = "El obstáculo SI intersecta con la horizontal
interna.";
                    hintalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 1]) + "m.";
                }
                if (this.intersecciones[this.posvector, 2] == false)
                {
                    aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                    apalt.Text = "";
                }
                else
                {
                    aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                    apalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 2]) + "m.";
                }
                if (this.intersecciones[this.posvector, 3] == false)
```

```
                {
                    transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                    transalt.Text = "";
                }
                else
                {
                    transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                    transalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 3]) + "m.";
                }
                if (this.intersecciones[this.posvector, 4] == false)
                {
                    SAIinfo.Text = "El obstáculo NO intersecta con la SAI.";
                    SAIalt.Text = "";
                }
                else
                {
                    SAIinfo.Text = "El obstáculo SI intersecta con la SAI.";
                    SAIalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 4]) + "m.";
                }
                if (this.intersecciones[this.posvector, 5] == false)
                {
                    acsdinfo.Text = "El obstáculo NO intersecta con la superficie
ACSD.";
                    ascdalt.Text = "";
                }
                else
                {
                    acsdinfo.Text = "El obstáculo SI intersecta con la superficie
ASCD.";
                    ascdalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 5]) + "m.";
                }
                aintinfo.Text = "--";
                transintinfo.Text = "--";
                X.Text = "X:" + Convert.ToString(this.pos[this.posvector, 0]);
                Y.Text = "Y:" + Convert.ToString(this.pos[this.posvector, 1]);
                Z.Text = "Z:" + Convert.ToString(this.pos[this.posvector, 2]);
            }
            else
            {
                if (this.intersecciones[this.posvector, 0] == false)
                {
                    coninfo.Text = "El obstáculo NO intersecta con la cónica.";
                    conalt.Text = "";
                }
                else
                {
                    coninfo.Text = "El obstáculo SI intersecta con la cónica.";
                    conalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 0]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 1] == false)
                {
                    Hintinfo.Text = "El obstáculo NO intersecta con la horizontal
interna.";
                    hintalt.Text = "";
                }
```

```
                else
                {
                    Hintinfo.Text = "El obstáculo SI intersecta con la horizontal
interna.";
                    hintalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 1]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 2] == false)
                {
                    aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                    apalt.Text = "";
                }
                else
                {
                    aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                    apalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 2]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 3] == false)
                {
                    aintinfo.Text = "El obstáculo NO intersecta con la superficie
de Ap.interna.";
                    apialt.Text = "";
                }
                else
                {
                    aintinfo.Text = "El obstáculo SI intersecta con la superficie
de Ap.interna.";
                    apialt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 3]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 4] == false)
                {
                    transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                    transalt.Text = "";
                }
                else
                {
                    transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                    transalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 4]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 5] == false)
                {
                    transintinfo.Text = "El obstáculo NO intersecta con la
superficie de T.interna.";
                    transintalt.Text = "";
                }
                else
                {
                    transintinfo.Text = "El obstáculo SI intersecta con la
superficie de T.interna.";
```

```
                    transintalt.Text = "La intersección se produce a una altura
de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector,
5]), 2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 6] == false)
                {
                    SAIinfo.Text = "El obstáculo NO intersecta con la SAI.";
                    SAIalt.Text = "";
                }
                else
                {
                    SAIinfo.Text = "El obstáculo SI intersecta con la SAI.";
                    SAIalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 6]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 7] == false)
                {
                    acsdinfo.Text = "El obstáculo NO intersecta con la superficie
ACSD.";
                    ascdalt.Text = "";
                }
                else
                {
                    acsdinfo.Text = "El obstáculo SI intersecta con la superficie
ASCD.";
                    ascdalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 7]),
2)) + "m.";
                }
                X.Text = "X:" + Convert.ToString(this.pos[this.posvector, 0]);
                Y.Text = "Y:" + Convert.ToString(this.pos[this.posvector, 1]);
                Z.Text = "Z:" + Convert.ToString(this.pos[this.posvector, 2]);
            }
        }

        private void anteriorbuttom_Click(object sender, EventArgs e)
        {
            if (this.posvector == 0)
            {
                this.posvector = (this.pos.Length / 3 - 1);
            }
            else
            {
                this.posvector = this.posvector - 1;
            }
            if (this.aproxval == "Visual" || this.aproxval == "NoPrec")
            {
                if (this.intersecciones[this.posvector, 0] == false)
                {
                    coninfo.Text = "El obstáculo NO intersecta con la cónica.";
                    conalt.Text = "";
                }
                else
                {
                    coninfo.Text = "El obstáculo SI intersecta con la cónica.";
                    conalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 0]) + "m.";
                }
                if (this.intersecciones[this.posvector, 1] == false)
                {
```

```
                    Hintinfo.Text = "El obstáculo NO intersecta con la horizontal
interna.";
                    hintalt.Text = "";
                }
                else
                {
                    Hintinfo.Text = "El obstáculo SI intersecta con la horizontal
interna.";
                    hintalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 1]) + "m.";
                }
                if (this.intersecciones[this.posvector, 2] == false)
                {
                    aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                    apalt.Text = "";
                }
                else
                {
                    aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                    apalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 2]) + "m.";
                }
                if (this.intersecciones[this.posvector, 3] == false)
                {
                    transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                    transalt.Text = "";
                }
                else
                {
                    transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                    transalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 3]) + "m.";
                }
                if (this.intersecciones[this.posvector, 4] == false)
                {
                    SAIinfo.Text = "El obstáculo NO intersecta con la SAI.";
                    SAIalt.Text = "";
                }
                else
                {
                    SAIinfo.Text = "El obstáculo SI intersecta con la SAI.";
                    SAIalt.Text = "La intersección se produce a una altura de:" +
Convert.ToString(this.alt[this.posvector, 4]) + "m.";
                }
                if (this.intersecciones[this.posvector, 5] == false)
                {
                    acsdinfo.Text = "El obstáculo NO intersecta con la superficie
ACSD.";
                    ascdalt.Text = "";
                }
                else
                {
                    acsdinfo.Text = "El obstáculo SI intersecta con la superficie
ASCD.";
                    ascdalt.Text = "La intersección se produce a una altura de:"
+ Convert.ToString(this.alt[this.posvector, 5]) + "m.";
                }
                aintinfo.Text = "--";
```

```
                transintinfo.Text = "--";
                X.Text = "X:" + Convert.ToString(this.pos[this.posvector, 0]);
                Y.Text = "Y:" + Convert.ToString(this.pos[this.posvector, 1]);
                Z.Text = "Z:" + Convert.ToString(this.pos[this.posvector, 2]);
            }
            else
            {
                if (this.intersecciones[this.posvector, 0] == false)
                {
                    coninfo.Text = "El obstáculo NO intersecta con la cónica.";
                    conalt.Text = "";
                }
                else
                {
                    coninfo.Text = "El obstáculo SI intersecta con la cónica.";
                    conalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 0]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 1] == false)
                {
                    Hintinfo.Text = "El obstáculo NO intersecta con la horizontal
interna.";
                    hintalt.Text = "";
                }
                else
                {
                    Hintinfo.Text = "El obstáculo SI intersecta con la horizontal
interna.";
                    hintalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 1]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 2] == false)
                {
                    aproxinfo.Text = "El obstáculo NO intersecta con la
superficie de aproximación.";
                    apalt.Text = "";
                }
                else
                {
                    aproxinfo.Text = "El obstáculo SI intersecta con la
superficie de aproximación.";
                    apalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 2]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 3] == false)
                {
                    aintinfo.Text = "El obstáculo NO intersecta con la superficie
de Ap.interna.";
                    apialt.Text = "";
                }
                else
                {
                    aintinfo.Text = "El obstáculo SI intersecta con la superficie
de Ap.interna.";
                    apialt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 3]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 4] == false)
```

```
                {
                    transinfo.Text = "El obstáculo NO intersecta con la
superficie de transición.";
                    transalt.Text = "";
                }
                else
                {
                    transinfo.Text = "El obstáculo SI intersecta con la
superficie de transición.";
                    transalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 4]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 5] == false)
                {
                    transintinfo.Text = "El obstáculo NO intersecta con la
superficie de T.interna.";
                    transintalt.Text = "";
                }
                else
                {
                    transintinfo.Text = "El obstáculo SI intersecta con la
superficie de T.interna.";
                    transintalt.Text = "La intersección se produce a una altura
de " + Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector,
5]), 2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 6] == false)
                {
                    SAIinfo.Text = "El obstáculo NO intersecta con la SAI.";
                    SAIalt.Text = "";
                }
                else
                {
                    SAIinfo.Text = "El obstáculo SI intersecta con la SAI.";
                    SAIalt.Text = "La intersección se produce a una altura de " +
Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 6]),
2)) + "m.";
                }
                if (this.intersecciones[this.posvector, 7] == false)
                {
                    acsdinfo.Text = "El obstáculo NO intersecta con la superficie
ACSD.";
                    ascdalt.Text = "";
                }
                else
                {
                    acsdinfo.Text = "El obstáculo SI intersecta con la superficie
ASCD.";
                    ascdalt.Text = "La intersección se produce a una altura de "
+ Convert.ToString(Decimal.Round(Convert.ToDecimal(this.alt[this.posvector, 7]),
2)) + "m.";
                }
                X.Text = "X:" + Convert.ToString(this.pos[this.posvector, 0]);
                Y.Text = "Y:" + Convert.ToString(this.pos[this.posvector, 1]);
                Z.Text = "Z:" + Convert.ToString(this.pos[this.posvector, 2]);
            }
        }
        public OBS[] GetObstlist()
        {
            return this.Oblist;
        }
```

```
        public bool Getvisible()
        {
            return this.visible[0];
        }
        //Método que borrará los valores que estén mostrandose en el panel de
datos del obstáculo siempre que se haya actualizado la lista de osbtáculos
        public void vaciarpanel()
        {
            X.Text = "X:";
            Y.Text = "Y:";
            Z.Text = "Z:";
            acsdinfo.Text = "--";
            ascdalt.Text = "";
            SAIinfo.Text = "--";
            SAIalt.Text = "";
            transintinfo.Text = "--";
            transintalt.Text = "";
            transinfo.Text = "--";
            transalt.Text = "";
            aintinfo.Text = "--";
            apialt.Text = "";
            aproxinfo.Text = "--";
            apalt.Text = "";
            Hintinfo.Text = "--";
            hintalt.Text = "";
            coninfo.Text = "--";
            conalt.Text = "";
        }

        private void manualmenteToolStripMenuItem_Click(object sender, EventArg
e)
        {
            Obstaculos vnt2 = new Obstaculos();
            vnt2.ShowDialog();
            visible[0] = vnt2.Getrellenado();
            if (visible[0] == true)
            {
                this.Oblist = vnt2.Getobs();
                this.pos = new double[this.Oblist.Length, 3];
                for (int i = 0; i < this.Oblist.Length; i++)
                {
                    this.pos[i, 0] = this.Oblist[i].Getpos()[0];
                    this.pos[i, 1] = this.Oblist[i].Getpos()[1];
                    this.pos[i, 2] = this.Oblist[i].Getpos()[2];
                }
                if (pos.Length / 3 > 1)
                {
                    visible[1] = true;
                }
                else
                {
                    visible[1] = false;
                }
                siguientebuttom.Visible = visible[1];
                anteriorbuttom.Visible = visible[1];
            }

        }
    }
}
```

Obstaculos.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LibreriaDimPen;

namespace LibreriaIntersecciones
{
    public partial class Obstaculos : Form
    {
        //Variable que indicará el número de obstáculos
        int num;
        //Clase en la cual se guardarán los obstáculos
        OBS[] Obslist;
        //Matriz en la cual se irán guardando las coordenadas de los obstáculos
        double[,] posi;
        //Vector en el cual se irán guardando la identificación de los obstáculos
        string[] id = new string[999];
        bool rellenado;
        bool mdu=false;
        bool error = false;
        public Obstaculos()
        {
            InitializeComponent();
            if (num == 0)
            {
                rellenado = false;
            }
            else
            {
                rellenado = true;
            }
        }

        private void Aceptar_Click(object sender, EventArgs e)
        {
            if (obs.RowCount >= 1 && (obs[0,0].Value != null && obs[1, 0].Value
!= null && obs[2, 0].Value != null))
            {
                try
                {
                    if (obs.RowCount == 2)
                    {
                        this.Obslist = new OBS[1];
                        this.posi = new double[1, 3];
                        this.posi[0, 0] =
Convert.ToDouble(Convert.ToString(obs[1, 0].Value));
                        this.posi[0, 1] =
Convert.ToDouble(Convert.ToString(obs[2, 0].Value));
                        this.posi[0, 2] =
Convert.ToDouble(Convert.ToString(obs[3, 0].Value));
                        this.id[0] = Convert.ToString(obs[0, 0].Value);
                        double[] pos = new double[3];
                        pos[0] = this.posi[0, 0];
                        pos[1] = this.posi[0, 1];
                        pos[2] = this.posi[0, 2];
```

```
                        OBS o = new OBS();
                        o.Setinfo(this.id[0], pos);
                        this.Obslist[0] = o;
                    }
                    else
                    {
                        this.num = obs.RowCount - 1;
                        //Primero se revisará si existe algún obstáculo con la
misma identificación que otro, si es el caso se descartará la fila
                        //siempre se descartá la información que haya sido
introducida más tarde
                        //Si se ha encintrado ya uno igual sale del segundo bucle
                        bool igual = false;
                        for (int i = 1; i < this.num; i++)
                        {
                            if (igual == true)
                            {
                                i = 1;
                            }
                            //Variable d que nos servirá para comparar
                            string dd = Convert.ToString(obs[0, this.num -
i].Value); igual = false;
                            for (int j = 1 + i; (j <= this.num) && igual ==
false; j++)
                            {
                                if (dd == Convert.ToString(obs[0, this.num -
j].Value))
                                {
                                    obs.Rows.RemoveAt(this.num - i);
                                    this.num = this.num - 1;
                                    igual = true;
                                }
                            }
                        }
                        //Matriz temporal
                        double [,] m = new double[this.num, 3];
                        //Vector temporal
                        string[] d = new string[this.num];
                        //Contador para filas vacías
                        int v = 0;
                        for (int i = 0; i < this.num; i++)
                        {
                            if (obs[0, i].Value != null && obs[1, i].Value !=
null && obs[2, i].Value != null && obs[3,i]!=null)
                            {
                                d[i - v] = Convert.ToString(obs[0, i].Value);
                                m[i-v, 0] =
Convert.ToDouble(Convert.ToString(obs[1, i].Value));
                                m[i-v, 1] =
Convert.ToDouble(Convert.ToString(obs[2, i].Value));
                                m[i-v, 2] =
Convert.ToDouble(Convert.ToString(obs[3, i].Value));
                            }
                            else
                            {
                                v = v + 1;
                            }
                        }
                        this.posi = new double[(m.Length / 3)-v, 3];
                        this.Obslist = new OBS[(m.Length / 3) - v];
                        if (v != 0)
                        {
```

```
                            for (int i = 0; i < (this.posi.Length / 3); i++)
                            {
                                this.posi[i, 0] = m[i, 0];
                                this.posi[i, 1] = m[i, 1];
                                this.posi[i, 2] = m[i, 2];
                                double[] pos = new double[3];
                                pos[0] = this.posi[i, 0];
                                pos[1] = this.posi[i, 1];
                                pos[2] = this.posi[i, 2];
                                this.id[i] = d[i];
                                OBS o = new OBS();
                                o.Setinfo(this.id[i], pos);
                                this.Obslist[i] = o;
                            }
                            rellenar(this.Obslist);
                        }
                        else
                        {
                            this.posi = m;
                            this.id = d;
                            for (int i = 0; i < (this.posi.Length / 3); i++)
                            {
                                double[] pos = new double[3];
                                pos[0] = this.posi[i, 0];
                                pos[1] = this.posi[i, 1];
                                pos[2] = this.posi[i, 2];
                                OBS o = new OBS();
                                o.Setinfo(this.id[i], pos);
                                this.Obslist[i] = o;
                            }
                        }
                        mdu = true;
                    }
                    rellenado = true;
                }
                catch(Exception a)
                {
                    MessageBox.Show("" + a.Message);
                }
            }
            else
            {
                MessageBox.Show("Por favor, introduzca al menos las coordenadas
de 1 obstáculo.");
            }
        }
        //Método que rellena una lista de obstáculos
        public void Setinfo(OBS[] Obslist)
        {
            this.Obslist = new OBS[Obslist.Length];
            this.Obslist = Obslist;
        }
        //Método que reestablece todos los valores y vacía el datagridview
        public void reset()
        {
            this.posi = null;
            vaciar();
            rellenado = false;
            this.num = 0;
            this.Obslist = new OBS[1];
        }
```

```
        //Método que rellena el datagridview con los datos dde una lista de
obstáculos
        public void rellenar(OBS[] Obslist)
        {
            vaciar();
            this.obs.Rows.Add((Obslist.Length));
            for (int i = 0; i < (Obslist.Length); i++)
            {
                double[] pos = Obslist[i].Getpos();
                this.obs[0, i].Value = Convert.ToString(Obslist[i].Getid());
                this.obs[1, i].Value = Convert.ToString(pos[0]);
                this.obs[2, i].Value = Convert.ToString(pos[1]);
                this.obs[3, i].Value = Convert.ToString(pos[2]);
            }
            rellenado = true;
        }
        //Método que elimina todos los datos que haya en el datagridview
        public void vaciar()
        {
            int j = this.obs.RowCount - 1;
            for (int i = 0; i < j; i++)
            {
                this.obs.Rows.RemoveAt(0);
            }
        }
        //Cerrar
        private void Cancelar_Click(object sender, EventArgs e)
        {
            Close();
        }
        //Opción reestablecer
        private void reestablecerToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            reset();
        }
        //MÉTODOS GET

        public bool Getrellenado()
        {
            return this.rellenado;
        }
        public OBS[] Getobs()
        {
            return this.Obslist;
        }

        private void cerrarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void Eliminar_Click(object sender, EventArgs e)
        {
            if(this.obs.RowCount == 2 && (this.obs[0,1].Value==null))
            {
                rellenado = false;
            }
            if (this.obs.RowCount == 1)
            {
                if (this.obs[0, 0].Value == null)
                {
```

```
                    MessageBox.Show("No hay más datos que eliminar");
                }
                rellenado = false;
            }
            else
            {
                //Variable que indicará la fila eliminada para actualizar la
lista de instalaciones
                int eliminado;
                //Variable que indicará las filas eliminadas;
                int v = 0;
                this.num = obs.RowCount - 1;
                double[,] temppos = new double[this.num, 3];
                foreach (DataGridViewCell oneCell in obs.SelectedCells)
                {
                    if (oneCell.Selected)
                    {
                        eliminado = oneCell.RowIndex;
                        obs.Rows.RemoveAt(oneCell.RowIndex);
                        v = v + 1;
                    }
                }
                this.posi = new double[(this.obs.RowCount) - 1, 3];
                this.Obslist=new OBS[(this.obs.RowCount) - 1];
                if (this.obs.RowCount >= 1 && this.obs[0, 0].Value != null)
                {
                    for (int i = 0; i < (this.obs.RowCount) - 1; i++)
                    {
                        this.posi[i, 0] =
Convert.ToDouble(Convert.ToString(obs[1, i].Value));
                        this.posi[i, 1] =
Convert.ToDouble(Convert.ToString(obs[2, i].Value));
                        this.posi[i, 2] =
Convert.ToDouble(Convert.ToString(obs[3, i].Value));
                        double[] pos = new double[3];
                        pos[0] = this.posi[i, 0];
                        pos[1] = this.posi[i, 1];
                        pos[2] = this.posi[i, 2];
                        this.id[i] = Convert.ToString(obs[0, i].Value);
                        OBS o = new OBS();
                        o.Setinfo(this.id[i], pos);
                        this.Obslist[i] = o;
                    }
                    mdu = true;
                    rellenado = true;
                }
                else
                {
                    this.posi = new double[1, 3];
                    this.posi = null;
                }
            }
        }


    }

}
```

Form.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LibreriaDimPen;
using DibujarSuperficies;
using LibreriaIntersecciones;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Interop.Common;
using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.DatabaseServices;
using System.Runtime.InteropServices;
using System.Data.OleDb;


namespace Menu
{
    public partial class Form1 : Form
    {
        //Variable que indica la longitud de la pista
        double longitudpista;
        //Variable que indica el tipo de aproximación
        string aproxval;
        //Variable quer indica la categoría de la pista
        string catletra;
        //Variable que indicará el ancho de la pista
        int anchopista;
        //Variable que indicará los metros desplazados del umbral
        double posumbral;
        //Variable que servirá para guardar las carcterísticas dimensionales de
las SLO
        DimPen dp = new DimPen();
        //Variable que servirá para guardar las características dimensionales de
la franja
        Franjadim fjd = new Franjadim();
        //Variable que indicará la altura de la OCA
        double OCA;
        //Vector booleano, los valores del cual irán variando según si se han
introducido correctamente los valores y así permitir que se depure correctamente
el código
        bool[] OK = {false,false,false};
        //Variable que indicará la orientación de aterrizaje
        double aterrizaje;
        //Variable que indicará la orientación de despegue
        double despegue;
        //Variable que contendrá la distancia que abarca la zona libre de
obstáculos
        double cwy;
        //Variable booleano que indicará si se han consultado los valores de
manera correcta
        bool consult = false;
        //Variable que indica el número de clave de la pista
        int clave=0;
```

```
        //Matriz que contiene los obstáculos (identificación y coordenadas)
        double[,] posi;
        //Matriz que contiene las coordenadas de la pista
        double[,] pistapos;
        //Vector que contiene la escala del mapa
        double[] valoresc;
        //Variable que indicará el sentido del aterrizaje
        double sentaterrizaje;
        //Variable que indicará el sentido de despegue
        double sentdespegue;
        //Variable que guardará el ángulo de valor mínimo de una de las cabeceras
respecto el norte
        double ang;
        //Variable que indicará si se han introducido los obstáculos
        bool obst = false;
        //Variable que contendrá la identificación y las coordenadas de los
obstáculos
        OBS[] Obslist;
        //Variable que contendrá las instalaciones radioeléctricas
        SRad[] srad;
        //Variable que indircará el número de sup radioeléctricas(variable temp)
        int sradnum = 0;
        //Ángulo con el cual produciremos el giro
        double angg;
        public Form1()
        {
            InitializeComponent();
        }

        //
        private void Consultar_Click(object sender, EventArgs e)
        {
            //Si todo ha sido rellenado
            if (valorlong.Text != "" && aprox.SelectedItem != "" &&
categoletra.SelectedItem != "" && (PredOCA.Checked==true || OCABox.Text!="") &&
(Extremo.Checked==true || (Desplazado.Checked==true && (DesplazadoBox.Text!="")))
&& AterrizajeBox.Text!="" && DespegueBox.Text!="" && cwyBox.Text!="")
            {
                try
                {
                    //Se intentará pasar los valores introducidos a las variables
                    longitudpista = Convert.ToDouble(valorlong.Text);
                    object apx = aprox.SelectedItem;
                    object catl=categoletra.SelectedItem;
                    aterrizaje = Convert.ToDouble(AterrizajeBox.Text);
                    despegue = Convert.ToDouble(DespegueBox.Text);
                    cwy = Convert.ToDouble(cwyBox.Text);
                    pistapos = new double[1, 3];
                    pistapos[0,0]=Convert.ToDouble(xBox.Text);
                    pistapos[0,1]=Convert.ToDouble(yBox.Text);
                    pistapos[0,2]=Convert.ToDouble(zBox.Text);
                    if (Extremo.Checked == true)
                    {
                        //Al estar en el extremo el valor será 0
                        posumbral = 0;
                    }
                    else
                    {
                        posumbral = Convert.ToDouble(DesplazadoBox.Text);
                    }
                    if (PredOCA.Checked == true)
                    {
```

```
                        //Valores predefinidos por normativa
                        OCA = 150;
                        //Todo es correcto así que el primer valor de OK cambia a
true
                        OK [0]= true;
                    }
                    else
                    {
                        if (Convert.ToDouble(OCABox.Text) < 150)
                        {
                            MessageBox.Show("La normativa recomienda que el valor
de la OCA tiene que ser como mínimo de 150m, por favor inserte un valor igual o
superior a 150m");
                        }
                        else
                        {
                            OCA = Convert.ToDouble(OCABox.Text);
                            OK[0] = true;
                        }
                    }
                    valoresc = new double[2];
                    valoresc[0] = Convert.ToDouble(eBox1.Text);
                    valoresc[1] = Convert.ToDouble(eBox2.Text);
                    if (valoresc[0] == 0 || valoresc[1] == 0 || valoresc[0] < 0
|| valoresc[1] < 0)
                    {
                        MessageBox.Show("El valor de la escala no puede ser igual
o menor a 0");
                    }
                    else
                    {
                        OK[2] = true;
                    }
                    //Si todos los datos han sido guardados correctamente se
puede pasar a la segunda fase de la consulta llamando a la función Consult
                    if (OK[0] == true && OK[2]==true)
                    {
                        Consult(longitudpista, apx, catl, OCA);
                    }
                }
                catch
                {
                    MessageBox.Show("Por favor introduzca la información con el
formato correcto. ");
                }
            }
            else
            {
                MessageBox.Show("¡Por favor, rellene y seleccione todos los
campos!");
            }
        }
        public void Consult(double longitudpista,object apx, object catl, double
OCA)
        {
            try
            {
                catletra=Convert.ToString(catl);
                //Preparamos las variables para consultar
                if (apx == "Aproximación de precisión CAT II o III")
                {
                    aproxval = "Prec23";
```

```
                }
                if (apx == "Aproximación de precisión CAT I")
                {
                    aproxval = "Prec1";
                }
                if (apx == "Aproximación que no sea de precisión")
                {
                    aproxval = "Noprec";
                }
                if (apx == "Aproximación visual")
                {
                    aproxval = "Visual";
                }
            }
            catch
            {
                MessageBox.Show("Por favor, seleccione una de las opciones de
aproximación.");
            }
            //Condiciones que se pondrán de acuerdo a la normativa.
            if (longitudpista<1200)
            {
                if(catletra=="A" || catletra=="B" || catletra=="C")
                {
                    if(longitudpista<800)
                    {
                        clave=1;
                        if(catletra=="A" || catletra=="B")
                        {
                            anchopista=18;
                        }
                        else
                        {
                            anchopista=23;
                        }
                    }
                    else
                        clave=2;
                        {
                        if(catletra=="A" || catletra=="B")
                        {
                            anchopista=23;
                        }
                        else
                        {
                            anchopista=30;
                        }
                    }
                }
                else
                {
                    MessageBox.Show("Para pistas de clave 1 o 2(longitud inferior
a 800m y longitud entre 800m-1200m respectivamente) la letra de la pista solo
puede ser A,B o C");
                    categoletra.Text = null;
                }
            }
            if (longitudpista >= 1200 && longitudpista < 1800)
            {
                clave = 3;
                if (catletra == "E" || catletra == "F")
                {
```

```
                    MessageBox.Show("Para pistas de clave 3(longitud entre 1200m-
1800m) la letra de la pista solo puede ser A,B,C o D");
                    categoletra.Text = null;
                }
                else
                {
                    if (catletra == "A"||catletra == "B"||catletra == "C")
                    {
                        anchopista=30;
                    }
                    if (catletra == "D")
                    {
                        anchopista=45;
                    }
                }
            }
            if (longitudpista > 1800)
            {
                clave = 4;
                if (catletra == "A" || catletra == "B")
                {
                    MessageBox.Show("Para pistas de clave 4(longitud superior
1800m) la letra de la pista solo puede ser E y F");
                    categoletra.Text = null;
                }
                else
                {
                    if (catletra == "C" || catletra == "D" || catletra == "E")
                    {
                        anchopista = 45;
                    }
                    if (catletra == "F")
                    {
                        anchopista = 60;
                    }
                }
            }
            if (categoletra.SelectedItem != null)
            {
                double penascd = 2;
                bool noche=false;
                if (longitudpista >= 1200)
                {
                    if (aproxval == "Visual" || aproxval == "Noprec")
                    {
                        while (OK[1] == false)
                        {
                            Noche vnt2 = new Noche();
                            vnt2.ShowDialog();
                            bool OKn = vnt2.GetOKn();
                            if (OKn == true)
                            {
                                penascd = vnt2.Getpenascd();
                                noche = vnt2.Getnoche();
                                OK[1] = true;
                            }
                            else
                            {
                                MessageBox.Show("Por favor, seleccione una opción
para cada condición.");
                            }
                        }
```

```
                    }
                    else
                    {
                        while (OK[1] == false)
                        {
                            Pendientedesp vnt2 = new Pendientedesp();
                            vnt2.ShowDialog();
                            bool OKn = vnt2.GetOKn();
                            if (OKn == true)
                            {
                                penascd = vnt2.Getpenascd();
                                OK[1] = true;
                            }
                            else
                            {
                                MessageBox.Show("Por favor, seleccione una
opción.");
                            }
                        }
                    }
                }
                else
                {
                    //Si se cumpren todas las condiciones ya podemos consultar
los valores
                    OK[1] = true;
                }
                if (OK[1] == true)
                {
                    if (longitudpista < 1200 && apx == "Aproximación de precisión
CAT II o III")
                    {
                        valorlong.Text = null;
                        aprox.Text = null;
                        MessageBox.Show("Para aproximaciones de precisión de CAT
II o III es necesario una pista con longitud igual o superior a 1200 m.");
                        reset();
                    }
                    else
                    {
                        //Si todo está en orden se guardarán los valores
dimensionales en la variable dp y se rellenarán los labels del Form con los
valores buscados.
                        fjd = fjd.GetFranjadim(aproxval, longitudpista);
                        dp = dp.GetDimPen(longitudpista, aproxval, catletra, OCA,
fjd,cwy, noche,penascd);
                        pencon.Text = "Pendiente:" +
Convert.ToString(dp.Getpencon());
                        altcon.Text = "Altura:" +
Convert.ToString(dp.Getaltcon());
                        radhint.Text = "Radio:" +
Convert.ToString(dp.Getradhint());
                        althint.Text = "Altura:" +
Convert.ToString(dp.Getalthint());
                        if (dp.Getanchapint() == 0)
                        {
                            anchapi.Text = "Anchura:--";
                        }
                        else
                        {
                            anchapi.Text = "Anchura:" +
Convert.ToString(dp.Getanchapint());
```

```
                        }
                        if (dp.Getdistuaint() == 0)
                        {
                            distuapi.Text = "Distancia desde el umbral:--";
                        }
                        else
                        {
                            distuapi.Text = "Distancia desde el umbral:" +
Convert.ToString(dp.Getdistuaint());
                        }
                        if (dp.Getlongaint() == 0)
                        {
                            longapi.Text = "Longitud:--";
                        }
                        else
                        {
                            longapi.Text = "Longitud:" +
Convert.ToString(dp.Getlongaint());
                        }
                        if (dp.Getpenaint() == 0)
                        {
                            penapi.Text = "Pendiente:";
                        }
                        else
                        {
                            penapi.Text = "Pendiente:" +
Convert.ToString(dp.Getpenaint());
                        }
                        if (dp.Getpenaint() == 0)
                        {
                            penapi.Text = "Pendiente:--";
                        }
                        else
                        {
                            penapi.Text = "Pendiente:" +
Convert.ToString(dp.Getpenaint());
                        }
                        longbiap.Text = "Long. Borde interior:" +
Convert.ToString(dp.Getlongbiap());
                        distuap.Text = "Disancia desde el umbral:" +
Convert.ToString(dp.Getdistuap());
                        divap.Text = "Divergencia (a cada lado):" +
Convert.ToString(dp.Getdivap());
                        pseclong.Text = "Longitud:" +
Convert.ToString(dp.Getpseclong());
                        psecpen.Text = "Pendiente:" +
Convert.ToString(dp.Getpsecpen());
                        if (dp.Getsseclong() == 0)
                        {
                            sseclong.Text = "Longitud:--";
                        }
                        else
                        {
                            sseclong.Text = "Longitud:" +
Convert.ToString(dp.Getsseclong());
                        }
                        if (dp.Getssecpen() == 0)
                        {
                            ssecpen.Text = "Pendiente:--";
                        }
                        else
                        {
```

```
                            ssecpen.Text = "Pendiente:" +
Convert.ToString(dp.Getssecpen());
                        }
                        if (dp.Getshorlong() == 0)
                        {
                            shorlong.Text = "Longitud:--";
                        }
                        else
                        {
                            shorlong.Text = "Longitud:" +
Convert.ToString(dp.Getshorlong());
                        }
                        if (dp.Getshorlongt() == 0)
                        {
                            shorlongt.Text = "Longitud total:--";
                        }
                        else
                        {
                            shorlongt.Text = "Longitud total:" +
Convert.ToString(dp.Getshorlongt());
                        }
                        pentrans.Text = "Pendiente:" +
Convert.ToString(dp.Getpentrans());
                        if (dp.Getpentransint() == 0)
                        {
                            pentransint.Text = "Pendiente:--";
                        }
                        else
                        {
                            pentransint.Text = "Pendiente:" +
Convert.ToString(dp.Getpentransint());
                        }
                        if (dp.Getlongbisai() == 0)
                        {
                            longbisai.Text = "Long. Borde interior:--";
                        }
                        else
                        {
                            longbisai.Text = "Long. Borde interior:" +
Convert.ToString(dp.Getlongbisai());
                        }
                        if (dp.Getdistusai() == 0)
                        {
                            distusai.Text = "Distancia desde el unbral:--";
                        }
                        else
                        {
                            distusai.Text = "Distancia desde el umbral:" +
Convert.ToString(dp.Getdistusai());
                        }
                        if (dp.Getdivsai() == 0)
                        {
                            divsai.Text = "Divergencia (a cada lado):--";
                        }
                        else
                        {
                            divsai.Text = "Divergencia (a cada lado):" +
Convert.ToString(dp.Getdivsai());
                        }
                        if (dp.Getpensai() == 0)
                        {
                            pensai.Text = "Pendiente:--";
```

```
                        }
                        else
                        {
                            pensai.Text = "Pendiente:" +
Convert.ToString(dp.Getpensai());
                        }
                        LongbiAscd.Text = "Long. Borde interior:" +
Convert.ToString(dp.Getlongbiascd());
                        DistextpisAcsd.Text = "Distancia desde el extremo de
pista:" + Convert.ToString(dp.Getdistextpascd());
                        DivAscd.Text = "Divergencia (a cada lado):" +
Convert.ToString(dp.Getdivascd());
                        LongAscd.Text = "Longitud:" +
Convert.ToString(dp.Getlongascd());
                        AnchAscd.Text = "Anchura final:" +
Convert.ToString(dp.Getanchascd());
                        PenAscd.Text = "Pendiente:" +
Convert.ToString(dp.Getpenascd());
                        //Reiniciamos los valores de OK para poder realizar
futuras consultas
                        OK[0] = false;
                        OK[1] = false;
                        OK[2] = false;
                        MessageBox.Show("Proceso de consulta completa.");
                        consult = true;
                        //Definimos el sentido de aterrizaje y despegue
                        if (this.aterrizaje < this.despegue)
                        {
                            this.sentaterrizaje = 1;
                            this.sentdespegue = -1;
                            if (this.aterrizaje == 90)
                            {
                                this.ang = 0;
                            }
                            else
                            {
                                    if (this.aterrizaje < 90)
                                    {
                                        this.ang = -(90 - this.aterrizaje);
                                    }
                                    else
                                    {
                                        this.ang = this.aterrizaje - 90;
                                    }
                            }
                        }
                        else
                        {
                            this.sentaterrizaje = -1;
                            this.sentdespegue = 1;
                            if (this.despegue == 90)
                            {
                                this.ang = 0;
                            }
                            else
                            {
                                    if (this.despegue < 90)
                                    {
                                        this.ang = -(90 - this.despegue);
                                    }
                                    else
                                    {
```

```
                                        this.ang = this.despegue - 90;
                                    }
                                }
                            }
                            //Determinamos la orientación de los letreros, de las
alturas(z) de las curvas de nivel
                            if (this.ang == 0)
                            {
                                this.angg = 0;
                            }
                            else
                            {
                                if (this.ang < 0)
                                {
                                    this.angg = 90 - this.ang;
                                }
                                else
                                {
                                    this.angg = this.ang + 90;
                                }
                            }
                        }
                    }
                    else
                    {
                    }
                }
                else
                {
                    MessageBox.Show("Por favor para continuar, rellene correctamente
todos los campos.");
                }
            }
            private void button2_Click(object sender, EventArgs e)
            {
                reset();
            }
            public void reset()
            {
                //Se reestablecen los valores
                valorlong.Text = null;
                aprox.Text = null;
                categoletra.Text = null;
                DimPen dp = new DimPen();
                longitudpista = 0;
                anchopista = 0;
                Extremo.Checked = false;
                Desplazado.Checked = false;
                DesplazadoBox.Text = null;
                PredOCA.Checked = false;
                OCABox.Text = null;
                aproxval = null;
                AterrizajeBox.Text = null;
                DespegueBox.Text = null;
                cwyBox.Text = null;
                OK[0] = false;
                OK[1] = false;
                consult = false;
                clave = 0;
                this.posi = null;
                this.valoresc = null;
                this.ang = 0;
```

```
            pencon.Text = "Pendiente:";
            altcon.Text = "Altura:";
            radhint.Text = "Radio:";
            althint.Text = "Altura:";
            anchapi.Text = "Anchura:";
            distuapi.Text = "Distancia desde el umbral:";
            longapi.Text = "Longitud:";
            penapi.Text = "Pendiente:";
            longbiap.Text = "Long. Borde interior:";
            distuapi.Text = "Distacia desde el umbral:";
            divap.Text = "Divergencia (a cada lado):";
            pseclong.Text = "Longitud:";
            psecpen.Text = "Pendiente:";
            sseclong.Text = "Longitud:";
            ssecpen.Text = "Pendiente:";
            shorlong.Text = "Longitud:";
            shorlongt.Text = "Longitud total:";
            pentrans.Text = "Pendiente:";
            pentransint.Text = "Pendiente:";
            longbisai.Text = "Long. Borde interior:";
            distusai.Text = "Distancia desde el umbral:";
            divsai.Text = "Divergencia (a cada lado):";
            pensai.Text = "Pendiente:";
            LongbiAscd.Text = "Long. Borde interior:";
            DistextpisAcsd.Text = "Distancia desde el extremo de pista:";
            DivAscd.Text = "Divergencia (a cada lado):";
            LongAscd.Text = "Longitud:";
            AnchAscd.Text = "Anchura final:";
            PenAscd.Text = "Pendiente:";
            eBox1.Text = null;
            eBox2.Text = null;
            xBox.Text = null;
            yBox.Text = null;
            zBox.Text = null;
        }

        private void dibujar_Click(object sender, EventArgs e)
        {
            //Comprobación de que todos los datos se han introducido
correctamente, si es asi
            //se ejecutará AUTOCAD y se comenzará a representar las servidumbres
            if (consult == true)
            {
                Dibuclass dibu = new Dibuclass();
                Intersecciones its = new Intersecciones();
                double[] pospista = new double[3];
                pospista[0] = this.pistapos[0, 0];
                pospista[1] = this.pistapos[0, 1];
                pospista[2] = this.pistapos[0, 2];
                dibu.Setinfo(this.dp, its, this.valoresc, this.ang, aproxval,
Convert.ToDouble(valorlong.Text), pospista,
                    this.sentaterrizaje, this.sentdespegue, this.posumbral,
this.clave, this.fjd, this.anchopista);
                pospista = its.giro(pospista, this.ang);
                AcadApplication acApp = null;
                try
                {
                    acApp =
(AcadApplication)Marshal.GetActiveObject("AutoCAD.Application");
                }
                catch
                {
```

```
                    try
                    {
                        Type acType =
Type.GetTypeFromProgID("AutoCAD.Application");
                        acApp = (AcadApplication)Activator.CreateInstance(acType,
true);
                    }
                    catch
                    {
                        MessageBox.Show("Cannot create object of type \"" +
"AutoCAD.Application" + "\"");
                    }
                }
                if (acApp != null)
                {
                    // By the time this is reached AutoCAD is fully
                    // functional and can be interacted with through code

                    acApp.Visible = true;
                    //Representación de la pista
                    acApp.ActiveDocument.SendCommand("-color 252 ");
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    double[,] pista = new double[4, 2];
                    pista = dibu.Getpista();
                    acApp.ActiveDocument.SendCommand("._pline "
                       + Convert.ToString(pista[0, 0]).Replace(",", ".") + "," +
Convert.ToString(pista[0, 1]).Replace(",", ".") + " "
                       + Convert.ToString(pista[1, 0]).Replace(",", ".") + "," +
Convert.ToString(pista[1, 1]).Replace(",", ".") + " "
                       + Convert.ToString(pista[2, 0]).Replace(",", ".") + "," +
Convert.ToString(pista[2, 1]).Replace(",", ".") + " "
                       + Convert.ToString(pista[3, 0]).Replace(",", ".") + "," +
Convert.ToString(pista[3, 1]).Replace(",", ".") + " "
                       + Convert.ToString(pista[0, 0]).Replace(",", ".") + "," +
Convert.ToString(pista[0, 1]).Replace(",", ".") + " " + " ");
                    //A continuación se representarán las SLO en AutoCAD.

                    //HORIZONTAL INTERNA.
                    double[,] Hip = new double[6, 2];
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    Hip = dibu.GetHIp();
                    acApp.ActiveDocument.SendCommand("-color 80 ");
                    acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(Hip[2, 0]).Replace(",", ".") + "," + Convert.ToString(Hip[2,
1]).Replace(",", ".") + " "
                       + Convert.ToString(Hip[0, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[0, 1]).Replace(",", ".") + " "
                       + Convert.ToString(Hip[1, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[1, 1]).Replace(",", ".") + " ");
                    acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(Hip[4, 0]).Replace(",", ".") + "," + Convert.ToString(Hip[4,
1]).Replace(",", ".") + " "
                       + Convert.ToString(Hip[3, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[3, 1]).Replace(",", ".") + " "
                       + Convert.ToString(Hip[5, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[5, 1]).Replace(",", ".") + " ");
                    acApp.ActiveDocument.SendCommand("._line "
                       + Convert.ToString(Hip[1, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[1, 1]).Replace(",", ".") + " "
                       + Convert.ToString(Hip[4, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[4, 1]).Replace(",", ".") + " " + " ");
                    acApp.ActiveDocument.SendCommand("._line "
```

```
                        + Convert.ToString(Hip[5, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[5, 1]).Replace(",", ".") + " "
                        + Convert.ToString(Hip[2, 0]).Replace(",", ".") + "," +
Convert.ToString(Hip[2, 1]).Replace(",", ".") + " " + " ");

                    //CÓNICA.
                    double[,] CON = new double[6, 2];
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    CON = dibu.Getcon();
                    acApp.ActiveDocument.SendCommand("-color 80 ");
                    acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(CON[2, 0]).Replace(",", ".") + "," + Convert.ToString(CON[2,
1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[0, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[0, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[1, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[1, 1]).Replace(",", ".") + " ");
                    acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(CON[4, 0]).Replace(",", ".") + "," + Convert.ToString(CON[4,
1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[3, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[3, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[5, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[5, 1]).Replace(",", ".") + " ");
                    acApp.ActiveDocument.SendCommand("._line "
                        + Convert.ToString(CON[1, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[1, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[4, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[4, 1]).Replace(",", ".") + " " + " ");
                    acApp.ActiveDocument.SendCommand("._line "
                        + Convert.ToString(CON[5, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[5, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[2, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[2, 1]).Replace(",", ".") + " " + " ");
                    //Representamos las curvas de nivel
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                    for (int i = 6; i < CON.Length / 2; i = i + 6)
                    {
                        acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(CON[i + 2, 0]).Replace(",", ".") + "," + Convert.ToString(CON[i
+ 2, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[i, 0]).Replace(",", ".") + "," +
Convert.ToString(CON[i, 1]).Replace(",", ".") + " "
                        + Convert.ToString(CON[i + 1, 0]).Replace(",", ".") + ","
+ Convert.ToString(CON[i + 1, 1]).Replace(",", ".") + " ");
                        acApp.ActiveDocument.SendCommand("._arc " +
Convert.ToString(CON[i + 4, 0]).Replace(",", ".") + "," + Convert.ToString(CON[i
+ 4, 1]).Replace(",", ".") + " "
                            + Convert.ToString(CON[i + 3, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 3, 1]).Replace(",", ".") + " "
                            + Convert.ToString(CON[i + 5, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 5, 1]).Replace(",", ".") + " ");
                        acApp.ActiveDocument.SendCommand("._line "
                            + Convert.ToString(CON[i + 1, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 1, 1]).Replace(",", ".") + " "
                            + Convert.ToString(CON[i + 4, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 4, 1]).Replace(",", ".") + " " + " ");
                        acApp.ActiveDocument.SendCommand("._line "
                            + Convert.ToString(CON[i + 5, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 5, 1]).Replace(",", ".") + " "
                            + Convert.ToString(CON[i + 2, 0]).Replace(",", ".") +
"," + Convert.ToString(CON[i + 2, 1]).Replace(",", ".") + " " + " ");
```

```
                    }
                    //Representamos las alturas(z) de las curvas de nivel
                    double[,] TextCON = new double[dibu.Gettext().Length, 3];
                    TextCON = dibu.Gettext();
                    for (int i = 0; i < TextCON.Length / 3; i++)
                    {
                        acApp.ActiveDocument.SendCommand("-texto "
                        + Convert.ToString(TextCON[i, 1]).Replace(",", ".") + ","
+ Convert.ToString(TextCON[i, 2]).Replace(",", ".") + " "
                        + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                        + Convert.ToString(this.angg).Replace(",", ".") + " "
                        + Convert.ToString(TextCON[i, 0]) + "\n");
                    }

                    //APROXIMACIÓN
                    double[,] AP = new double[4, 2];
                    AP = dibu.Getap();
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    acApp.ActiveDocument.SendCommand("-color 10 ");
                    acApp.ActiveDocument.SendCommand("._pline "
                        + Convert.ToString(AP[0, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[0, 1]).Replace(",", ".") + " "
                        + Convert.ToString(AP[1, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[1, 1]).Replace(",", ".") + " "
                        + Convert.ToString(AP[3, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[3, 1]).Replace(",", ".") + " "
                        + Convert.ToString(AP[2, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[2, 1]).Replace(",", ".") + " "
                        + Convert.ToString(AP[0, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[0, 1]).Replace(",", ".") + " " + " ");
                    //Representamos las curvas de nivel
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                    for (int i = 4; i < AP.Length / 2; i = i + 2)
                    {
                        acApp.ActiveDocument.SendCommand("._line "
                        + Convert.ToString(AP[i, 0]).Replace(",", ".") + "," +
Convert.ToString(AP[i, 1]).Replace(",", ".") + " "
                        + Convert.ToString(AP[i + 1, 0]).Replace(",", ".") + ","
+ Convert.ToString(AP[i + 1, 1]).Replace(",", ".") + " " + " ");
                    }
                    //Representamos las alturas(z) de las curvas de nivel
                    double[,] TextAP = new double[dibu.Gettext().Length, 3];
                    TextAP = dibu.Gettext();
                    for (int i = 0; i < TextAP.Length / 3; i++)
                    {
                        acApp.ActiveDocument.SendCommand("-texto "
                        + Convert.ToString(TextAP[i, 1]).Replace(",", ".") + ","
+ Convert.ToString(TextAP[i, 2]).Replace(",", ".") + " "
                        + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                        + Convert.ToString(this.angg).Replace(",", ".") + " "
                        + Convert.ToString(TextAP[i, 0]) + "\n");
                    }

                    ////TRANSICIÖN
                    double[,] TR = new double[8, 2];
                    TR = dibu.GetTR();
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    acApp.ActiveDocument.SendCommand("-color 10 ");
                    acApp.ActiveDocument.SendCommand("._pline "
```

```
                        + Convert.ToString(TR[0, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[0, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[1, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[1, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[3, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[3, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[2, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[2, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[0, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[0, 1]).Replace(",", ".") + " " + " ");
                    acApp.ActiveDocument.SendCommand("._pline "
                        + Convert.ToString(TR[4, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[4, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[5, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[5, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[6, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[6, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[7, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[7, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[4, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[4, 1]).Replace(",", ".") + " " + " ");
                    //Representamos las curvas de nivel
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                    for (int i = 8; i < TR.Length / 2; i = i + 2)
                    {
                        acApp.ActiveDocument.SendCommand("._line "
                        + Convert.ToString(TR[i, 0]).Replace(",", ".") + "," +
Convert.ToString(TR[i, 1]).Replace(",", ".") + " "
                        + Convert.ToString(TR[i + 1, 0]).Replace(",", ".") + ","
+ Convert.ToString(TR[i + 1, 1]).Replace(",", ".") + " " + " ");
                    }
                    //Representamos las alturas(z) de las curvas de nivel
                    double[,] TextTR = new double[dibu.Gettext().Length, 3];
                    TextTR = dibu.Gettext();
                    for (int i = 0; i < TextTR.Length / 3; i++)
                    {
                        acApp.ActiveDocument.SendCommand("-texto "
                        + Convert.ToString(TextTR[i, 1]).Replace(",", ".") + ","
+ Convert.ToString(TextTR[i, 2]).Replace(",", ".") + " "
                        + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                        + Convert.ToString(this.angg).Replace(",", ".") + " "
                        + Convert.ToString(TextTR[i, 0]) + "\n");
                    }


                    //ASCD
                    acApp.ActiveDocument.SendCommand("-color 170 ");
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    double[,] ASCD = new double[dibu.GetASCD().Length / 2, 2];
                    int j = 0;
                    if (this.longitudpista >= 1200)
                    {
                        ASCD = dibu.GetASCD();
                        acApp.ActiveDocument.SendCommand("._pline "
                            + Convert.ToString(ASCD[0, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[0, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[1, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[1, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[2, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[2, 1]).Replace(",", ".") + " "
```

```
                            + Convert.ToString(ASCD[3, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[3, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[4, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[4, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[5, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[5, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[0, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[0, 1]).Replace(",", ".") + " " + " ");
                        j = 6;
                    }
                    else
                    {
                        ASCD = dibu.GetASCD();
                        acApp.ActiveDocument.SendCommand("._pline "
                            + Convert.ToString(ASCD[0, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[0, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[1, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[1, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[3, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[3, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[2, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[2, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[0, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[0, 1]).Replace(",", ".") + " " + " ");
                        j = 4;
                    }
                    //Representamos las curvas de nivel
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                    for (int i = j; i < ASCD.Length / 2; i = i + 2)
                    {
                        acApp.ActiveDocument.SendCommand("._line "
                        + Convert.ToString(ASCD[i, 0]).Replace(",", ".") + "," +
Convert.ToString(ASCD[i, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ASCD[i + 1, 0]).Replace(",", ".") +
"," + Convert.ToString(ASCD[i + 1, 1]).Replace(",", ".") + " " + " ");
                    }
                    //Representamos las alturas(z) de las curvas de nivel
                    double[,] TextASCD = new double[dibu.Gettext().Length, 3];
                    TextASCD = dibu.Gettext();
                    for (int i = 0; i < TextASCD.Length / 3; i++)
                    {
                        acApp.ActiveDocument.SendCommand("-texto "
                        + Convert.ToString(TextASCD[i, 1]).Replace(",", ".") +
"," + Convert.ToString(TextASCD[i, 2]).Replace(",", ".") + " "
                            + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                            + Convert.ToString(this.angg).Replace(",", ".") + " "
                            + Convert.ToString(TextASCD[i, 0]) + "\n");
                    }

                    if (this.aproxval == "Prec1" || this.aproxval == "Prec23")
                    {
                    //Aproximación interna
                    double[,] ApI = new double[dibu.GetApI().Length / 2, 2];
                    ApI = dibu.GetApI();
                    acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                    acApp.ActiveDocument.SendCommand("-color 30 ");
                    acApp.ActiveDocument.SendCommand("._pline "
                        + Convert.ToString(ApI[0, 0]).Replace(",", ".") + "," +
Convert.ToString(ApI[0, 1]).Replace(",", ".") + " "
                        + Convert.ToString(ApI[1, 0]).Replace(",", ".") + "," +
Convert.ToString(ApI[1, 1]).Replace(",", ".") + " "
```

```
                        + Convert.ToString(ApI[3, 0]).Replace(",", ".") + "," +
Convert.ToString(ApI[3, 1]).Replace(",", ".") + " "
                        + Convert.ToString(ApI[2, 0]).Replace(",", ".") + "," +
Convert.ToString(ApI[2, 1]).Replace(",", ".") + " "
                        + Convert.ToString(ApI[0, 0]).Replace(",", ".") + "," +
Convert.ToString(ApI[0, 1]).Replace(",", ".") + " " + " ");
                        //Representamos las curvas de nivel
                        acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                        for (int i = 4; i < ApI.Length / 2; i = i + 2)
                        {
                            acApp.ActiveDocument.SendCommand("._line "
                            + Convert.ToString(ApI[i, 0]).Replace(",", ".") + ","
+ Convert.ToString(ApI[i, 1]).Replace(",", ".") + " "
                            + Convert.ToString(ApI[i + 1, 0]).Replace(",", ".") +
"," + Convert.ToString(ApI[i + 1, 1]).Replace(",", ".") + " " + " ");
                        }
                        //Representamos las alturas(z) de las curvas de nivel
                        double[,] TextApI = new double[dibu.Gettext().Length, 3];
                        TextApI = dibu.Gettext();
                        for (int i = 0; i < TextApI.Length / 3; i++)
                        {
                            acApp.ActiveDocument.SendCommand("-texto "
                            + Convert.ToString(TextApI[i, 1]).Replace(",", ".") +
"," + Convert.ToString(TextApI[i, 2]).Replace(",", ".") + " "
                            + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                            + Convert.ToString(this.angg).Replace(",", ".") + " "
                            + Convert.ToString(TextApI[i, 0]) + "\n");
                        }

                        //Transición interna
                        double[,] TrI = new double[dibu.GetTrI().Length / 2, 2];
                        TrI = dibu.GetTrI();
                        acApp.ActiveDocument.SendCommand("-color 200 ");
                        acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                        acApp.ActiveDocument.SendCommand("._pline "
                            + Convert.ToString(TrI[0, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[0, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[1, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[1, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[2, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[2, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[3, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[3, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[4, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[4, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[0, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[0, 1]).Replace(",", ".") + " " + " ");
                        acApp.ActiveDocument.SendCommand("._pline "
                            + Convert.ToString(TrI[5, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[5, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[6, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[6, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[7, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[7, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[8, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[8, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[9, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[9, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[5, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[5, 1]).Replace(",", ".") + " " + " ");
                        //Representamos las curvas de nivel
```

```
                        acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                        for (int i = 10; i < TrI.Length / 2; i = i + 2)
                        {
                            acApp.ActiveDocument.SendCommand("._line "
                            + Convert.ToString(TrI[i, 0]).Replace(",", ".") + ","
+ Convert.ToString(TrI[i, 1]).Replace(",", ".") + " "
                            + Convert.ToString(TrI[i + 1, 0]).Replace(",", ".") +
"," + Convert.ToString(TrI[i + 1, 1]).Replace(",", ".") + " " + " ");
                        }
                        //Representamos las alturas(z) de las curvas de nivel
                        double[,] TextTrI = new double[dibu.Gettext().Length, 3];
                        TextTrI = dibu.Gettext();
                        for (int i = 0; i < TextTrI.Length / 3; i++)
                        {
                            acApp.ActiveDocument.SendCommand("-texto "
                            + Convert.ToString(TextTrI[i, 1]).Replace(",", ".") +
"," + Convert.ToString(TextTrI[i, 2]).Replace(",", ".") + " "
                            + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                            + Convert.ToString(this.angg).Replace(",", ".") + " "
                            + Convert.ToString(TextTrI[i, 0]) + "\n");
                        }

                        //SAI
                        double[,] SAI = new double[4, 2];
                        SAI = dibu.GetSAI();
                        acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                        acApp.ActiveDocument.SendCommand("-color 104 ");
                        acApp.ActiveDocument.SendCommand("._pline "
                            + Convert.ToString(SAI[0, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[0, 1]).Replace(",", ".") + " "
                            + Convert.ToString(SAI[1, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[1, 1]).Replace(",", ".") + " "
                            + Convert.ToString(SAI[3, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[3, 1]).Replace(",", ".") + " "
                            + Convert.ToString(SAI[2, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[2, 1]).Replace(",", ".") + " "
                            + Convert.ToString(SAI[0, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[0, 1]).Replace(",", ".") + " " + " ");
                        //Representamos las curvas de nivel
                        acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                        for (int i = 4; i < SAI.Length / 2; i = i + 2)
                        {
                            acApp.ActiveDocument.SendCommand("._line "
                            + Convert.ToString(SAI[i, 0]).Replace(",", ".") + ","
+ Convert.ToString(SAI[i, 1]).Replace(",", ".") + " "
                            + Convert.ToString(SAI[i + 1, 0]).Replace(",", ".") +
"," + Convert.ToString(SAI[i + 1, 1]).Replace(",", ".") + " " + " ");
                        }
                        //Representamos las alturas(z) de las curvas de nivel
                        double[,] TextSAI = new double[dibu.Gettext().Length, 3];
                        TextSAI = dibu.Gettext();
                        for (int i = 0; i < TextSAI.Length / 3; i++)
                        {
                            acApp.ActiveDocument.SendCommand("-texto "
                            + Convert.ToString(TextSAI[i, 1]).Replace(",", ".") +
"," + Convert.ToString(TextSAI[i, 2]).Replace(",", ".") + " "
                            + Convert.ToString(10 * this.valoresc[0] /
this.valoresc[1]).Replace(",", ".") + " "
                            + Convert.ToString(this.angg).Replace(",", ".") + " "
                            + Convert.ToString(TextSAI[i, 0]) + "\n");
                        }
```

```
                    }
                }
                MessageBox.Show("Servidumbres representadas correctamente.");
            }
            else
            {
                MessageBox.Show("Por favor, introduzca primero los parámetros de
la pista que definirán las superfícies limitadoras de obstáculos.");
            }
        }

        private void PredOCA_CheckedChanged(object sender, EventArgs e)
        {
            if (PredOCA.Checked == true)
            {
                OCABox.Text = null;
                PredOCA.Checked = true;
            }
        }

        private void OCABox_TextChanged(object sender, EventArgs e)
        {
            PredOCA.Checked = false;
        }

        private void Extremo_CheckedChanged(object sender, EventArgs e)
        {
            if (Extremo.Checked == true)
            {
                DesplazadoBox.Text = null;
                Extremo.Checked=true;
            }
        }

        private void DesplazadoBox_TextChanged(object sender, EventArgs e)
        {
            Extremo.Checked = false;
        }

        private void DespegueBox_TextChanged(object sender, EventArgs e)
        {
            try
            {
                if (DespegueBox.Text == "")
                {
                    AterrizajeBox.Text = null;
                }
                else
                {
                    despegue = Convert.ToDouble(DespegueBox.Text);
                    if (despegue >= 180)
                    {
                        aterrizaje = despegue - 180;
                        AterrizajeBox.Text = Convert.ToString(aterrizaje);
                    }
                    else
                    {
                        aterrizaje = despegue + 180;
                        AterrizajeBox.Text = Convert.ToString(aterrizaje);
                    }
                }
```

```
            }
            catch
            {
                    DespegueBox.Text == "";
            }

        }

        private void AterrizajeBox_TextChanged(object sender, EventArgs e)
        {
            try
            {
                if (AterrizajeBox.Text == "")
                {
                    DespegueBox.Text = null;
                }
                else
                {
                    aterrizaje = Convert.ToDouble(AterrizajeBox.Text);
                    if (aterrizaje >= 180)
                    {
                        despegue = aterrizaje - 180;
                        DespegueBox.Text = Convert.ToString(despegue);
                    }
                    else
                    {
                        despegue = aterrizaje + 180;
                        DespegueBox.Text = Convert.ToString(despegue);
                    }
                }
            }
            catch
            {
                    AterrizajeBox.Text == "";
            }
        }

        private void Intersecbuttom_Click_1(object sender, EventArgs e)
        {
            if (consult == true)
            {
                IntersecInfo vnt2 = new IntersecInfo();
                vnt2.Setinfo(valorlong.Text, clave, catletra, aprox.SelectedItem,
dp, pistapos, aproxval, sentaterrizaje, sentdespegue, posumbral, fjd, ang, obst,
Obslist);
                vnt2.ShowDialog();
                this.posi = vnt2.Getpos();
                this.obst = vnt2.Getvisible();
                if (this.obst == true)
                {
                    this.Obslist = vnt2.GetObstlist();
                }
                else
                {
                    this.Obslist = new OBS[1];
                }
            }
            else
            {
                MessageBox.Show("Por favor, introduzca primero los parámetros de
la pista que definirán las superfícies limitadoras de obstáculos.");
            }
```

```
        }
        public double[,] Getpos()
        {
            return this.pistapos;
        }

        private void SRadToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (consult == true)
            {
                if (this.sradnum == 0)
                {
                    this.srad = new SRad[1];
                }
                RAD vnt2 = new RAD();
                double[] pospista = new double[3];
                pospista[0] = this.pistapos[0, 0];
                pospista[1] = this.pistapos[0, 1];
                pospista[2] = this.pistapos[0, 2];
                vnt2.Setinfo(this.obst, this.Obslist, this.anchopista,
this.posumbral, pospista, this.longitudpista, this.sentaterrizaje, this.sradnum,
this.srad, this.valoresc, this.ang);
                vnt2.ShowDialog();
                this.obst = vnt2.Getobst();
                if (this.obst == true)
                {
                    this.Obslist = vnt2.GetObslist();
                }
                else
                {
                    this.Obslist = new OBS[1];
                }
                this.sradnum=vnt2.Getnum();
                if (this.sradnum == 0)
                {
                    this.srad = new SRad[1];
                }
                else
                {
                    this.srad = vnt2.GetSrad();
                }
            }
            else
            {
                MessageBox.Show("Por favor, introduzca primero los parámetros de
la pista que definirán las superfícies limitadoras de obstáculos.");
            }
        }
        private void consultarModificarObstáculosToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (this.obst == true)
            {
                Obstaculos vnt2 = new Obstaculos();
                vnt2.Setinfo(this.Obslist);
                vnt2.rellenar(this.Obslist);
                vnt2.ShowDialog();
                //Comprobamos si sigue habiendo obstáculos porque se puede haber
reiniciado la lista
                this.obst = vnt2.Getrellenado();
                if (this.obst == true)
                {
```

```
                    this.Obslist = vnt2.Getobs();
                }
                else
                {
                    this.Obslist = new OBS[1];
                }
            }
            else
            {
                MessageBox.Show("Por favor, cargue o introduzca antes las
coordenadas de los obstáculos.");
            }
        }

        private void introducirObstáculosToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Obstaculos vnt2 = new Obstaculos();
            vnt2.ShowDialog();
            this.obst = vnt2.Getrellenado();
            if (this.obst == true)
            {
                this.Obslist = vnt2.Getobs();
            }
        }

    }
}
```

Instalaciones.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using LibreriaDimPen;
using LibreriaIntersecciones;

namespace Menu
{
    public partial class Instalaciones : Form
    {
        int num;
        SRad [] srad;
        public Instalaciones()
        {
            InitializeComponent();
        }

        public void Setinfo(SRad[] srad)
        {
            this.srad = new SRad[srad.Length];
            this.srad = srad;
            rellenar(this.srad);
        }
        public void vaciar()
        {
            int j = this.inst.RowCount - 1;
            for (int i = 0; i < j; i++)
            {
                this.inst.Rows.RemoveAt(0);
            }
        }
        public void rellenar(SRad [] srad)
        {
            vaciar();
            this.inst.Rows.Add((srad.Length));
            for (int i = 0; i < (srad.Length); i++)
            {
                this.inst[0, i].Value = srad[i].Getid();
                this.inst[1, i].Value = srad[i].Getant();
                double [] pos=new double[3];
                pos=srad[i].Getradpos();
                this.inst[2, i].Value = Convert.ToString(pos[0]);
                this.inst[3, i].Value = Convert.ToString(pos[1]);
                this.inst[4, i].Value = Convert.ToString(pos[2]);
            }
        }

        private void Eliminar_Click(object sender, EventArgs e)
        {
            if (this.inst.RowCount == 1 && (this.inst[0,0].Value==null))
            {
                MessageBox.Show("No hay más datos que eliminar");
            }
            else
```

```
            {
                //Variable que indicará la fila eliminada para actualizar la
lista de instalaciones
                int eliminado;
                //Variable que indicará las filas eliminadas;
                int el = 0;
                SRad[] sradtemp = new SRad[this.srad.Length];
                sradtemp = this.srad;
                foreach (DataGridViewCell oneCell in inst.SelectedCells)
                {
                    if (oneCell.Selected)
                    {
                        eliminado = oneCell.RowIndex;
                        inst.Rows.RemoveAt(oneCell.RowIndex);
                        el = el + 1;
                        for (int j = 0; eliminado + j < sradtemp.Length -1; j++)
                        {
                            sradtemp[eliminado + j] = sradtemp[eliminado + j +
1];
                        }
                    }
                }
                this.num = this.srad.Length - el;
                this.srad = new SRad[this.srad.Length - el];
                if (this.srad.Length != 0)
                {
                    for (int i = 0; i < this.srad.Length; i++)
                    {
                        this.srad[i] = sradtemp[i];
                    }
                }
                else
                {
                    SRad s = new SRad();
                    this.srad = new SRad[1];
                    this.srad[0] = s;
                }
            }
        }

        private void Aceptar_Click(object sender, EventArgs e)
        {
            Close();
        }
        public SRad[] Getsrad()
        {
            return this.srad;
        }

        private void reset_Click(object sender, EventArgs e)
        {
            vaciar();
            SRad s = new SRad();
            this.num = 0;
            this.srad = new SRad[1];
            this.srad[0] = s;
        }
    }
}
```

InterRAD.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using LibreriaDimPen;
using DibujarSuperficies;
using LibreriaIntersecciones;
namespace Menu
{
    public partial class InterRAD : Form
    {
        bool visible = false;
        SRad[] srad;
        double[,] pos;
        //Variable que indicará la posición, del vector de servidumbres, de la
instalación que se esté mostrando en caso de que haya más de una instalación.
        int prad = 0;
        //Matriz que contendrá las intersecciones de los obstáculos con cada
instalación, cada fila mostrará las intersecciones de una instalación con todos
los
        //obstáculos(repartidos a lo largo de las columnas).
        bool[,] intersecc;
        //Matriz con las alturas de intersección
        string[,] interseccalt;
        //Variable con el cual llamaremos los métodos necesarios para definir si
existe o no intersecciones
        Intersecciones its = new Intersecciones();
        //Clase con las identificaciones y las coordenadas de los obstáculos
        OBS[] Obslist;
        //Variable de ancho de pista
        double ancho;
        //Variable con la posición del umbral
        double posumbral;
        //Variable con la posición de la pista
        double[] pistapos;
        //Variable con la longitud de la pista
        double longitudpista;
        //Sentido de aterrizaje
        double sentaterrizaje;
        //Ángulo de giro
        double ang;
        public InterRAD()
        {
            InitializeComponent();
            siguienterad.Visible = visible;
            anteriorrad.Visible = visible;
        }
        public void Setsrad(SRad[] srad)
        {
            this.srad = srad;
            if (this.srad.Length > 1)
            {
                this.visible = true;
            }
            siguienterad.Visible = visible;
```

```
            anteriorrad.Visible = visible;
            ID.Text = "Identificación:" + this.srad[0].Getid();
            insta.Text = "Tip.Instalación:" + this.srad[0].Gettipo();
            ant.Text = "Clase de antena:" + this.srad[0].Getant();
            double [] radpos=this.srad[0].Getradpos();
            xx.Text = "X:" + Convert.ToString(radpos[0]);
            yy.Text = "Y:" + Convert.ToString(radpos[1]);
            zz.Text = "Z:" + Convert.ToString(radpos[2]);
            if (this.srad[0].Getant() != "ILS/LOC" && this.srad[0].Getant() !=
"ILS/GP")
            {
                zseg.Text = "Zona de seguridad:" +
Convert.ToString(this.srad[0].Getzseg());
                zalt.Text = "Zona de limitación de alturas:" +
Convert.ToString(this.srad[0].Getzalt());
            }
            else
            {
                zseg.Text = "Zona de seguridad:";
                zalt.Text = "Zona de limitación de alturas:";
            }
            pen.Text = "Pendiente:" + Convert.ToString(this.srad[0].Getpen());
        }
        public void Setinfo(double ancho, double posumbral, double[] pistapos,
double longitudpista, double sentaterrizaje, double ang)
        {
            this.ancho = ancho;
            this.posumbral = posumbral;
            this.pistapos = pistapos;
            this.longitudpista = longitudpista;
            this.sentaterrizaje = sentaterrizaje;
            this.ang = ang;
        }
        public void Setpos(OBS[] Obslist)
        {
            vaciar();
            this.Obslist = new OBS[Obslist.Length];
            this.Obslist = Obslist;
            this.intersecc = new bool[this.srad.Length, this.Obslist.Length];
            this.interseccalt = new string[this.srad.Length,
this.Obslist.Length];
            this.obs.Rows.Add(this.Obslist.Length);
            pos = new double[Obslist.Length, 3];
            for (int i = 0; i < this.srad.Length; i++)
            {
                for (int j = 0; j < this.Obslist.Length; j++)
                {
                    pos[j, 0] = this.Obslist[j].Getpos()[0];
                    pos[j, 1] = this.Obslist[j].Getpos()[1];
                    pos[j, 2] = this.Obslist[j].Getpos()[2];
                    double[] temppos = new double[3];
                    temppos[0] = pos[j, 0];
                    temppos[1] = pos[j, 1];
                    temppos[2] = pos[j, 2];
                    temppos = its.giro(temppos, ang);
                    //Variable temporal con las posiciones de la instalación
                    double[] tempipos = new double[3];
                    tempipos[0] = this.srad[i].Getradpos()[0];
                    tempipos[1] = this.srad[i].Getradpos()[1];
                    tempipos[2] = this.srad[i].Getradpos()[2];
                    tempipos = its.giro(tempipos, ang);
```

```
                    if (this.srad[i].Getant() != "ILS/LOC" &&
this.srad[this.prad].Getant() != "ILS/GP")
                    {
                        this.intersecc[i, j] = its.radioint(this.srad[i],
temppos, this.srad[i].Getradpos());
                    }
                    else
                    {
                        if (this.srad[i].Getant() == "ILS/LOC")
                        {
                            this.intersecc[i, j] = its.ilocint(temppos, tempipos,
this.posumbral, this.sentaterrizaje, this.longitudpista, this.pistapos,this.ang);
                        }
                        else
                        {
                            this.intersecc[i, j] = its.igpint(temppos, tempipos,
this.posumbral, this.longitudpista, this.pistapos, this.ancho,this.ang);
                        }
                    }
                    if (this.intersecc[i, j] == true)
                    {
                        this.interseccalt[i, j] = Convert.ToString(its.Getalt());
                    }
                    else
                    {
                        this.interseccalt[i, j] = "--";
                    }
                    if (i == 0)
                    {
                        this.obs[0, j].Value = this.Obslist[j].Getid();
                        this.obs[1, j].Value = Convert.ToString(this.pos[j, 0]);
                        this.obs[2, j].Value = Convert.ToString(this.pos[j, 1]);
                        this.obs[3, j].Value = Convert.ToString(this.pos[j, 2]);
                        if (intersecc[i, j] == true)
                        {
                            this.obs[4, j].Value = "SI";
                        }
                        else
                        {
                            this.obs[4, j].Value = "NO";
                        }
                        this.obs[5, j].Value = this.interseccalt[i, j];
                    }
                }
            }
        }
        public void vaciar()
        {
            int j = this.obs.RowCount - 1;
            for (int i = 0; i < j; i++)
            {
                this.obs.Rows.RemoveAt(0);
            }
        }
        private void anteriorrad_Click(object sender, EventArgs e)
        {
            if (this.prad == 0)
            {
                this.prad = this.srad.Length - 1;
            }
            else
            {
```

```
                this.prad = this.prad - 1;
            }
            ID.Text = "Identificación:" + this.srad[this.prad].Getid();
            insta.Text = "Tip.Instalación:" + this.srad[this.prad].Gettipo();
            ant.Text = "Clase de antena:" + this.srad[this.prad].Getant();
            double[] radpos = this.srad[this.prad].Getradpos();
            xx.Text = "X:" + Convert.ToString(radpos[0]);
            yy.Text = "Y:" + Convert.ToString(radpos[1]);
            zz.Text = "Z:" + Convert.ToString(radpos[2]);
            if (this.srad[this.prad].Getant() != "ILS/LOC" &&
this.srad[this.prad].Getant() != "ILS/GP")
            {
                zseg.Text = "Zona de seguridad:" +
Convert.ToString(this.srad[this.prad].Getzseg());
                zalt.Text = "Zona de limitación de alturas:" +
Convert.ToString(this.srad[this.prad].Getzalt());
            }
            else
            {
                zseg.Text = "Zona de seguridad:";
                zalt.Text = "Zona de limitación de alturas:";
            }
            pen.Text = "Pendiente:" +
Convert.ToString(this.srad[this.prad].Getpen());
            for (int j = 0; j < this.pos.Length / 3; j++)
            {
                this.obs[1, j].Value = Convert.ToString(this.pos[j, 0]);
                this.obs[2, j].Value = Convert.ToString(this.pos[j, 1]);
                this.obs[3, j].Value = Convert.ToString(this.pos[j, 2]);
                if (intersecc[this.prad, j] == true)
                {
                    this.obs[4, j].Value = "SI";
                }
                else
                {
                    this.obs[4, j].Value = "NO";
                }
                this.obs[5, j].Value = this.interseccalt[this.prad, j];
            }
        }

        private void siguienterad_Click(object sender, EventArgs e)
        {
            if (this.prad == this.srad.Length-1)
            {
                this.prad = 0;
            }
            else
            {
                this.prad = this.prad + 1;
            }
            ID.Text = "Identificación:" + this.srad[this.prad].Getid();
            insta.Text = "Tip.Instalación:" + this.srad[this.prad].Gettipo();
            ant.Text = "Clase de antena:" + this.srad[this.prad].Getant();
            double[] radpos = this.srad[this.prad].Getradpos();
            xx.Text = "X:" + Convert.ToString(radpos[0]);
            yy.Text = "Y:" + Convert.ToString(radpos[1]);
            zz.Text = "Z:" + Convert.ToString(radpos[2]);
            if (this.srad[this.prad].Getant() != "ILS/LOC" &&
this.srad[this.prad].Getant() != "ILS/GP")
            {
```

```
                zseg.Text = "Zona de seguridad:" +
Convert.ToString(this.srad[this.prad].Getzseg());
                zalt.Text = "Zona de limitación de alturas:" +
Convert.ToString(this.srad[this.prad].Getzalt());
            }
            else
            {
                zseg.Text = "Zona de seguridad:";
                zalt.Text = "Zona de limitación de alturas:";
            }
            pen.Text = "Pendiente:" +
Convert.ToString(this.srad[this.prad].Getpen());
            for (int j = 0; j < this.pos.Length / 3; j++)
            {

                this.obs[1, j].Value = Convert.ToString(this.pos[j, 0]);
                this.obs[2, j].Value = Convert.ToString(this.pos[j, 1]);
                this.obs[3, j].Value = Convert.ToString(this.pos[j, 2]);
                if (intersecc[this.prad, j] == true)
                {
                    this.obs[4, j].Value = "SI";
                }
                else
                {
                    this.obs[4, j].Value = "NO";
                }
                this.obs[5, j].Value = this.interseccalt[this.prad, j];
            }
        }
        //Método que sirve para consultar/modificar la lista de obstáculos
        private void modificarrad_Click(object sender, EventArgs e)
        {
            Obstaculos vnt2 = new Obstaculos();
            vnt2.Setinfo(this.Obslist);
            vnt2.rellenar(this.Obslist);
            vnt2.ShowDialog();
            //En caso que el usuario haya eliminado toda la lista, tendrá que
introducir al menos 1 obstáculo para poder
            //ejecutar correcatmente el programe
            while (vnt2.Getrellenado() == false)
            {
                MessageBox.Show("Por favor, introduzca al menos un obstáculo
antes de continuar");
                vnt2.ShowDialog();
            }
            this.Obslist = vnt2.Getobs();
            //Variable temporal
            double[,] temppos = new double[this.Obslist.Length, 3];
            for (int i = 0; i < this.Obslist.Length; i++)
            {
                temppos[i, 0] = this.Obslist[i].Getpos()[0];
                temppos[i, 1] = this.Obslist[i].Getpos()[1];
                temppos[i, 2] = this.Obslist[i].Getpos()[2];
            }
            if (temppos != null)
            {
                this.pos = temppos;
            }
            Setpos(this.Obslist);
        }
        //Método que sirve para consultar/modificar la lista de instalaciones
        private void modificarinst_Click(object sender, EventArgs e)
```

```
        {
            RAD vnt2 = new RAD();
            vnt2.Visible();
            vnt2.SetSrad(this.srad);
            vnt2.ShowDialog();
            while (vnt2.Getnum() == 0)
            {
                MessageBox.Show("Por favor, introduzca al menos una instalación
antes de continuar");
                vnt2.ShowDialog();
            }
            this.srad = vnt2.GetSrad();
            if (this.srad.Length == 1)
            {
                visible = false;
            }
            else
            {
                visible = true;
            }

            siguienterad.Visible = visible;
            anteriorrad.Visible = visible;
            Setsrad(this.srad);
            Setpos(this.Obslist);

        }
        public OBS[] GetObslist()
        {
            return this.Obslist;
        }
        public SRad[] Getsrad()
        {
            return this.srad;
        }

        private void Cerrar_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Noche.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Menu
{
    public partial class Noche : Form
    {
        bool noche = false;
        double penascd = 2;
        bool OKn = false;
        public Noche()
        {
            InitializeComponent();
        }

        private void Aceptar_Click(object sender, EventArgs e)
        {
            if ((Si.Checked == true || No.Checked == true) && (Si2.Checked ==
true || No2.Checked == true))
            {
                if (Si.Checked == true)
                {
                    noche = true;
                }
                else
                {
                    noche = false;
                }
                if (Si2.Checked == true)
                {
                    Valorpen vp = new Valorpen();
                    vp.ShowDialog();
                    penascd = vp.Getpenascd();
                }
                OKn = true;
                Close();
            }
            else
            {
                MessageBox.Show("Por favor seleccione una opción para cada
condición.");
            }
        }
        public double Getpenascd()
        {
            return this.penascd;
        }
        public bool Getnoche()
        {
            return this.noche;
        }
        public bool GetOKn()
        {
            return this.OKn;
```

```
        }

        private void Si_CheckedChanged(object sender, EventArgs e)
        {
            if (Si.Checked == true)
            {
                No.Checked = false;
                Si.Checked = true;
            }
        }

        private void No_CheckedChanged(object sender, EventArgs e)
        {
            if (No.Checked == true)
            {
                Si.Checked = false;
                No.Checked = true;
            }
        }

        private void No2_CheckedChanged_1(object sender, EventArgs e)
        {
            if (No2.Checked == true)
            {
                Si2.Checked = false;
                No2.Checked = true;
            }
        }

        private void Si2_CheckedChanged_1(object sender, EventArgs e)
        {
            if (Si2.Checked == true)
            {
                No2.Checked = false;
                Si2.Checked = true;
            }
        }
    }
}
```

Pendientedesp.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Menu
{
    public partial class Pendientedesp : Form
    {
        bool OKn = false;
        double penascd = 2;

        public Pendientedesp()
        {
            InitializeComponent();
        }

        private void Aceptar_Click(object sender, EventArgs e)
        {
            if (Si.Checked == true || No.Checked == true)
            {
                if (Si.Checked == true)
                {
                    Valorpen vp = new Valorpen();
                    vp.ShowDialog();
                    penascd = vp.Getpenascd();
                }
                OKn = true;
                Close();
            }
            else
            {
                MessageBox.Show("Por favor seleccione una opción.");
            }
        }
        public double Getpenascd()
        {
            return this.penascd;
        }
        public bool GetOKn()
        {
            return this.OKn;
        }
    }
}
```

RAD.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using LibreriaDimPen;
using LibreriaIntersecciones;
using DibujarSuperficies;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Interop.Common;
using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.DatabaseServices;
using System.Runtime.InteropServices;

namespace Menu
{
    public partial class RAD : Form
    {
        //Definimos las variables que contendrán las características de las
servidumbres radioelélectrica de las diversas instalaciones.
        SRad srad = new SRad();
        SRad[] SRAD;
        SRad[] SRADtemp = new SRad[100];
        int num;
        //Vector de variables booleanos que indicarán que todo se ha rellenado
correctamente.
        bool []correcto = {false,false};
        string id;
        string ant;
        string tipo;
        double[] radposs = new double[3];
        //Matriz con las posiciones de los obstáculos
        double[,] pos;
        //Clase que contiene la identificación y las coordenadas de las
coordenadas
        OBS[] Obslist;
        //Variable de ancho de pista
        double ancho;
        //Variable con la posición del umbral
        double posumbral;
        //Variable con la posición de la pista
        double[] pistapos;
        //Variable con la longitud de la pista
        double longitudpista;
        //Sentido de aterrizaje
        double sentaterrizaje;
        //Variable que indicará si ya tenemos una lista de obstáculos
        bool obst;
       //Escala al que se querrá representar las servidumbres
        double[] escala = { 1, 1 };
        //Ángulo de rotación
        double ang;
        public RAD()
```

```
        {
            InitializeComponent();
        }

        private void CC_CheckedChanged(object sender, EventArgs e)
        {
            if (CC.Checked == true)
            {
                opciones.Items.Clear();
                opciones.Text = " ";
                opciones.Items.Add("Frecuencias bajas (LF) o medias (MF)");
                opciones.Items.Add("Frecuencias altas (HF)");
                opciones.Items.Add("Frecuencias muy altas (VHF) o ultra elevadas
(UHF)");
                CC.Checked = true;
            }
        }

        private void Rayuda_CheckedChanged(object sender, EventArgs e)
        {
            if (Rayuda.Checked == true)
            {
                opciones.Items.Clear();
                opciones.Text = " ";
                opciones.Items.Add("Radiobaliza marcadora de tipo 'Z' (75MHz)");
                opciones.Items.Add("Radiobaliza marcadora en abanico ('Fan
Marker') (75MHz)");
                opciones.Items.Add("Radiofaros no direccionales");
                opciones.Items.Add("Radiofaro omnidireccional VFH (VOR), equipo
medidor de distancia (DME) o TACAN");
                opciones.Items.Add("Radiogoniómetro VHF (VDF) o UHF (UDF)");
                opciones.Items.Add("Radar de vigilancia primario o secundario
(SSR)");
                opciones.Items.Add("ILS/LOC");
                opciones.Items.Add("ILS/GP");
                Rayuda.Checked = true;
            }
        }

        private void Introducir_Click(object sender, EventArgs e)
        {
            if (ID.Text != null && (CC.Checked == true || Rayuda.Checked == true)
&& opciones.SelectedItem != null)
            {
                try
                {
                    this.id = Convert.ToString(ID.Text);
                    this.ant = Convert.ToString(opciones.SelectedItem);
                    this.radposs[0] = Convert.ToDouble(XBox.Text);
                    this.radposs[1] = Convert.ToDouble(YBox.Text);
                    this.radposs[2] = Convert.ToDouble(ZBox.Text);
                    if (CC.Checked)
                    {
                        this.tipo = "cc";
                    }
                    else
                    {
                        this.tipo = "ray";
                    }
                    bool correcto = true;

                    if (correcto == true)
```

```
                    {
                        if (this.num == 0)
                        {
                            this.SRAD = new SRad[this.num + 1];
                            this.SRADtemp[this.num] =
this.srad.GetSRad(this.tipo, this.ant, this.id,this.radposs);
                            this.SRAD[0] = this.SRADtemp[this.num];
                            this.num = this.num + 1;
                            MessageBox.Show("Datos introducidos correctamente.");
                            ID.Text = null;
                            opciones.SelectedItem = null;
                            CC.Checked = false;
                            Rayuda.Checked = false;
                            XBox.Text = null;
                            YBox.Text = null;
                            ZBox.Text = null;
                        }
                        else
                        {
                            //Variable que indica si existe alguna instalación
con la misma identificación
                            bool igual=false;
                            for (int i = 0; (i < this.num) && (igual == false);
i++)
                            {
                                if (this.id == this.SRADtemp[i].Getid())
                                {
                                    igual = true;
                                }
                            }
                            //Si no existe ninguna coincidencia los datos se
cargan a la lista de instalaciones
                            if (igual == false)
                            {
                                this.SRADtemp[this.num] =
this.srad.GetSRad(this.tipo, this.ant, this.id, this.radposs);
                                this.num = this.num + 1;
                                this.SRAD = new SRad[this.num];
                                for (int i = 0; i < this.num; i++)
                                {
                                    this.SRAD[i] = this.SRADtemp[i];
                                }
                                MessageBox.Show("Datos introducidos
correctamente.");
                                ID.Text = null;
                                opciones.SelectedItem = null;
                                CC.Checked = false;
                                Rayuda.Checked = false;
                                XBox.Text = null;
                                YBox.Text = null;
                                ZBox.Text = null;
                            }
                            else
                            {
                                MessageBox.Show("La identificación introducida
coincide con la de otra instalación, por favor introduzca otra identificación.");
                            }
                        }
                    }
                }
                catch
                {
```

```
                    MessageBox.Show("Formato incorrecto.");
                }
            }
            else
            {
                MessageBox.Show("Por favor rellene correctamente todos los
datos.");
            }

        }

        private void consultarEliminarInstalaciónToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (this.num == 0)
            {
                MessageBox.Show("Por favor introduzca/defina primero al menos los
datos de una instalación.");
            }
            else
            {
                Instalaciones vnt2 = new Instalaciones();
                vnt2.Setinfo(this.SRAD);
                vnt2.ShowDialog();
                this.SRAD = vnt2.Getsrad();
                //En caso de que se hayan reestablecido los valores de las
instalaciones
                if (this.SRAD.Length == 1 && (this.SRAD[0].Getzseg() == 0 &&
this.SRAD[0].Getzalt() == 0 &&
                    this.SRAD[0].Getpen() == 0 && this.SRAD[0].Getid() == null &&
this.SRAD[0].Gettipo() == null
                    && this.SRAD[0].Getant() == null &&
this.SRAD[0].Getradpos()[0] == 0 && this.SRAD[0].Getradpos()[1] == 0 &&
this.SRAD[0].Getradpos()[2] == 0))
                {
                    this.num = 0;
                }
            }
        }

        private void Salir_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void determinarInterseccionesToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (this.num == 0)
            {
                MessageBox.Show("Por favor introduzca/defina primero al menos los
datos de una instalación.");
            }
            else
            {
                Obstaculos vnt2 = new Obstaculos();
                if (this.obst == false)
                {
                    vnt2.ShowDialog();
                    if (vnt2.Getrellenado() == true)
                    {
                        Intersecciones its = new Intersecciones();
```

```
                        //Variable temporal con la posición de la pista
transaladadas.
                        double[] temppistapos = new double[3];
                        temppistapos[0] = this.pistapos[0];
                        temppistapos[1] = this.pistapos[1];
                        temppistapos[2] = this.pistapos[2];
                        temppistapos = its.giro(temppistapos, this.ang);
                        this.Obslist = vnt2.Getobs();
                        InterRAD vnt3 = new InterRAD();
                        vnt3.Setsrad(this.SRAD);
                        vnt3.Setinfo(this.ancho, this.posumbral, temppistapos,
this.longitudpista, this.sentaterrizaje,this.ang);
                        vnt3.Setpos(this.Obslist);
                        vnt3.ShowDialog();
                        this.Obslist = vnt3.GetObslist();
                        this.SRAD = vnt3.Getsrad();
                    }
                    else
                    {
                        MessageBox.Show("Para determinar las intersecciones es
necesario que introduzca correctamente al menos 1 obstáculo.");
                    }
                }
                else
                {
                    vnt2.Setinfo(this.Obslist);
                    vnt2.rellenar(this.Obslist);
                    vnt2.ShowDialog();
                    //Comprobamos si sigue habiendo obstáculos porque se puede
haber reiniciado la lista
                    this.obst = vnt2.Getrellenado();
                    if (this.obst == true)
                    {
                        this.Obslist = vnt2.Getobs();
                        InterRAD vnt3 = new InterRAD();
                        vnt3.Setsrad(this.SRAD);
                        vnt3.Setpos(this.Obslist);
                        vnt3.ShowDialog();
                        this.Obslist = vnt3.GetObslist();
                        this.SRAD = vnt3.Getsrad();
                    }
                    else
                    {
                        this.Obslist = new OBS[1];
                        MessageBox.Show("Para determinar las intersecciones es
necesario que introduzca correctamente al menos 1 obstáculo.");
                    }
                }
            }
        }
        //Método que servirá para ocultar las opciones de Representar las
servidumbres e Determinar las intersecciones, de tal manera el usuario no
        //podrá utilizarlos cuando acceda a este form cuando intente modificar
las instalaciones desde el form InterRAD.
        public void Visible()
        {
            determinarInterseccionesToolStripMenuItem.Visible = false;
            representarServidumbresToolStripMenuItem.Visible = false;
        }
        //Métodos GET
        public SRad[] GetSrad()
        {
```

```
            return this.SRAD;
        }
        public int Getnum()
        {
            return this.num;
        }
        public void SetSrad(SRad[] srad)
        {
            this.SRAD = srad;
            this.num = this.SRAD.Length;
            for (int i = 0; i < this.num; i++)
            {
                this.SRADtemp[i] = srad[i];
            }
            for (int i = 0; i < SRAD.Length; i++)
            {
                this.SRADtemp[i] = this.SRAD[i];
            }
        }

        public bool Getobst()
        {
            return this.obst;
        }
        public OBS[] GetObslist()
        {
            return this.Obslist;
        }
        private void representarServidumbresToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Dibuclass dibu = new Dibuclass();
            if (this.num == 0)
            {
                MessageBox.Show("Por favor introduzca/defina primero al menos los
datos de una instalación.");
            }
            else
            {
                AcadApplication acApp = null;
                try
                {
                    acApp =
(AcadApplication)Marshal.GetActiveObject("AutoCAD.Application");
                }
                catch
                {
                    try
                    {
                        Type acType =
Type.GetTypeFromProgID("AutoCAD.Application");
                        acApp = (AcadApplication)Activator.CreateInstance(acType,
true);
                    }
                    catch
                    {
                        MessageBox.Show("Cannot create object of type \"" +
"AutoCAD.Application" + "\"");
                    }
                }
                if (acApp != null)
                {
```

```
                    acApp.Visible = true;
                    Intersecciones its = new Intersecciones();
                    DimPen dp = new DimPen();
                    Franjadim fjd=new Franjadim();
                    dibu.Setinfo(dp, its, this.escala, this.ang, " ",
this.longitudpista, this.pistapos, this.sentaterrizaje, this.sentaterrizaje * (-
1), this.posumbral, 0, fjd,this.ancho);
                    for (int i = 0; i < this.SRAD.Length; i++)
                    {
                        dibu.Setsrad(this.SRAD[i]);
                        if (this.SRAD[i].Getant() != "ILS/LOC" &&
this.SRAD[i].Getant() != "ILS/GP")
                        {
                            //Todas las superficies exceptuando ILS
                            double[] SRadI = new double[4];
                            SRadI = dibu.GetSRadE();
                            acApp.ActiveDocument.SendCommand("-color  139 ");
                            acApp.ActiveDocument.SendCommand("-grosorlin 0.3 ");
                            acApp.ActiveDocument.SendCommand("._circle " +
SRadI[0] + "," + SRadI[1] + ",0 " + SRadI[2] + " ");
                            acApp.ActiveDocument.SendCommand("._circle " +
SRadI[0] + "," + SRadI[1] + ",0 " + SRadI[3] + " ");
                            //Representamos las curvas de nivel
                            acApp.ActiveDocument.SendCommand("-grosorlin 0.25 ");
                            for (int j = 4; j < SRadI.Length; j++)
                            {
                                acApp.ActiveDocument.SendCommand("._circle " +
SRadI[0] + "," + SRadI[1] + ",0 " + SRadI[j] + " ");
                            }
                            //Representamos las alturas(z) de las curvas de nivel
                            double[,] TextCON = new double[dibu.Gettext().Length,
3];
                            TextCON = dibu.Gettext();
                            for (int j = 0; j < TextCON.Length / 3; j++)
                            {
                                acApp.ActiveDocument.SendCommand("-texto "
                                + Convert.ToString(TextCON[j, 1]).Replace(",",
".") + "," + Convert.ToString(TextCON[j, 2]).Replace(",", ".") + " "
                                + Convert.ToString(10 * this.escala[0] /
this.escala[1]).Replace(",", ".") + " "
                                + Convert.ToString(0).Replace(",", ".") + " "
                                + Convert.ToString(TextCON[j, 0]) + "\n");
                            }
                        }
                        else
                        {
                            if (this.SRAD[i].Getant() == "ILS/LOC")
                            {
                                acApp.ActiveDocument.SendCommand("-color 130 ");
                                acApp.ActiveDocument.SendCommand("-grosorlin 0.3
");
                                double[,] ILoc = new
double[dibu.GetILoc().Length, 2];
                                ILoc = dibu.GetILoc();
                                acApp.ActiveDocument.SendCommand("._pline "
                                    + Convert.ToString(ILoc[0, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[0, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[1, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[1, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[2, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[2, 1]).Replace(",", ".") + " "
```

```
                                    + Convert.ToString(ILoc[3, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[3, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[0, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[0, 1]).Replace(",", ".") + " " + " ");
                                acApp.ActiveDocument.SendCommand("._pline "
                                    + Convert.ToString(ILoc[2, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[2, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[7, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[7, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[6, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[6, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[1, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[1, 1]).Replace(",", ".") + " " + " ");
                                acApp.ActiveDocument.SendCommand("._pline "
                                    + Convert.ToString(ILoc[0, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[0, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[4, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[4, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[5, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[5, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[3, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[3, 1]).Replace(",", ".") + " " + " ");
                                acApp.ActiveDocument.SendCommand("._line "
                                    + Convert.ToString(ILoc[10, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[10, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[11, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[11, 1]).Replace(",", ".") + " " + " ");
                                acApp.ActiveDocument.SendCommand("._line "
                                    + Convert.ToString(ILoc[9, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[9, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[8, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[8, 1]).Replace(",", ".") + " " + " ");
                                //Representamos las curvas de nivel
                                acApp.ActiveDocument.SendCommand("-grosorlin 0.25
");

                                for (int k = 12; k < ILoc.Length / 2; k = k + 2)
                                {
                                    acApp.ActiveDocument.SendCommand("._line "
                                    + Convert.ToString(ILoc[k, 0]).Replace(",",
".") + "," + Convert.ToString(ILoc[k, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(ILoc[k + 1,
0]).Replace(",", ".") + "," + Convert.ToString(ILoc[k + 1, 1]).Replace(",", ".")
+ " " + " ");
                                }
                                //Representamos las alturas(z) de las curvas de
nivel
                                double[,] Textloc = new
double[dibu.Gettext().Length, 3];
                                Textloc = dibu.Gettext();
                                for (int k = 0; k < Textloc.Length / 3; k++)
                                {
                                    acApp.ActiveDocument.SendCommand("-texto "
                                    + Convert.ToString(Textloc[k,
1]).Replace(",", ".") + "," + Convert.ToString(Textloc[k, 2]).Replace(",", ".") +
" "
                                    + Convert.ToString(10 * this.escala[0] /
this.escala[1]).Replace(",", ".") + " "
                                    + Convert.ToString(0).Replace(",", ".") + " "
                                    + Convert.ToString(Textloc[k, 0]) + "\n");
                                }
                            }
                            else
```

```
                            {
                                acApp.ActiveDocument.SendCommand("-color 220 ");
                                acApp.ActiveDocument.SendCommand("-grosorlin 0.3
");
                                double[,] IGp = new double[dibu.GetIGp().Length,
2];
                                IGp = dibu.GetIGp();
                                acApp.ActiveDocument.SendCommand("._pline "
                                    + Convert.ToString(IGp[0, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[0, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[1, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[1, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[2, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[2, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[3, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[3, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[0, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[0, 1]).Replace(",", ".") + " " + " ");
                                acApp.ActiveDocument.SendCommand("._pline "
                                    + Convert.ToString(IGp[4, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[4, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[7, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[7, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[6, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[6, 1]).Replace(",", ".") + " "
                                    + Convert.ToString(IGp[5, 0]).Replace(",",
".") + "," + Convert.ToString(IGp[5, 1]).Replace(",", ".") + " " + " ");
                                //Representamos las curvas de nivel
                                acApp.ActiveDocument.SendCommand("-grosorlin 0.25
");
                                for (int j = 8; j < IGp.Length/2; j=j+2)
                                {
                                    acApp.ActiveDocument.SendCommand("._line "
                                        + Convert.ToString(IGp[j,
0]).Replace(",", ".") + "," + Convert.ToString(IGp[j, 1]).Replace(",", ".") + " "
                                        + Convert.ToString(IGp[j + 1,
0]).Replace(",", ".") + "," + Convert.ToString(IGp[j + 1, 1]).Replace(",", ".") +
" " + " ");
                                }
                                //Representamos las alturas(z) de las curvas de
nivel
                                double[,] Text = new
double[dibu.Gettext().Length, 3];
                                Text = dibu.Gettext();
                                for (int j = 0; j < Text.Length / 3; j++)
                                {
                                    acApp.ActiveDocument.SendCommand("-texto "
                                    + Convert.ToString(Text[j, 1]).Replace(",",
".") + "," + Convert.ToString(Text[j, 2]).Replace(",", ".") + " "
                                    + Convert.ToString(10 * this.escala[0] /
this.escala[1]).Replace(",", ".") + " "
                                    + Convert.ToString(0).Replace(",", ".") + " "
                                    + Convert.ToString(Text[j, 0]) + "\n");
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        public void Setinfo(bool obst, OBS[] Obslist, double ancho, double
posumbral, double[] pistapos, double longitudpista, double sentaterrizaje, int
num, SRad[]srad,double[] escala,double ang)
        {
            this.obst = obst;
            if (this.obst == true)
            {
                this.Obslist = new OBS[Obslist.Length];
                this.Obslist = Obslist;
            }
            this.ancho = ancho;
            this.posumbral = posumbral;
            this.pistapos = pistapos;
            this.longitudpista = longitudpista;
            this.sentaterrizaje = sentaterrizaje;
            this.num = num;
            if (this.num > 0)
            {
                for (int i = 0; i < this.num; i++)
                {
                    this.SRADtemp[i] = srad[i];
                }
                this.SRAD = srad;
            }
            this.escala = escala;
            this.ang = ang;
        }
    }
}
```

Valorpen.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Menu
{
    public partial class Valorpen : Form
    {
        double penascd = 2;
        public Valorpen()
        {
            InitializeComponent();
        }

        private void Aceptar_Click(object sender, EventArgs e)
        {
            if (penBox.Text == null)
            {
                MessageBox.Show("Por favor introduzca un valor.");
            }
            else
            {
                try
                {
                    penascd = Convert.ToDouble(penBox.Text);
                    Close();
                }
                catch (Exception a)
                {
                    MessageBox.Show("" + a.Message);
                }
            }
        }

        private void Cancelar_Click(object sender, EventArgs e)
        {
            penascd = 2;
            Close();
        }
        public double Getpenascd()
        {
            return this.penascd;
        }
    }
}
```